

新版 図解

# Z-80の使い方

横田 英一 著



オーム社



新版 図解

# Z-80の使い方

横田 英一 著



オーム社

---

本書は、「著作権法」によって、著作権等の権利が保護されている著作物です。本書の複製権・翻訳権・上映権・譲渡権・公衆送信権（送信可能化権を含む）は著作権者が保有しています。本書の全部または一部につき、無断で転載、複写複製、電子的装置への入力等をされると、著作権等の権利侵害となる場合がありますので、ご注意ください。

本書の無断複写は、著作権法上の制限事項を除き、禁じられています。本書の複写複製を希望される場合は、そのつど事前に下記へ連絡して許諾を得てください。

（株）日本著作出版権管理システム（電話 03-3817-5670、FAX 03-3815-8199）

---

**JCLS** <（株）日本著作出版権管理システム委託出版物>

---

## は し が き

---

「マイクロコンピュータ Z-80 の使い方」は、ちょうど世の中にマイクロコンピュータが普及しはじめ、エレクトロニクスエンジニアの新しいツールとして脚光を浴びはじめた時期に刊行し、なかでも本流となった Z-80 の解りやすい入門書として、10 万を越す読者を得、この本で育った先輩から、後輩へ受け継がれております。このような当初予想もしなかった反響に驚くとともに、強い責任をも感じている次第です。

エレクトロニクスの進化の速度は級数的に増し、マイクロコンピュータの世界も、16 ビットから 32 ビットへと広がり、要求と可能性はエスカレートし、RISC、並列処理などの新しいキーワードによる夢は限りなく拡大しております。しかし、いかに高性能なマイクロコンピュータといえども、ノイマン型であるかぎり基本は同じで、スキルの習得には一定のアプローチルールがあるように思えます。

8 ビットマイコンは汎用性にすぐれており、機能的によく整理され、性能において標準的な位置にあります。教材の入手も容易でマイクロコンピュータを習得するうえでのステップとしては、手頃なものです。

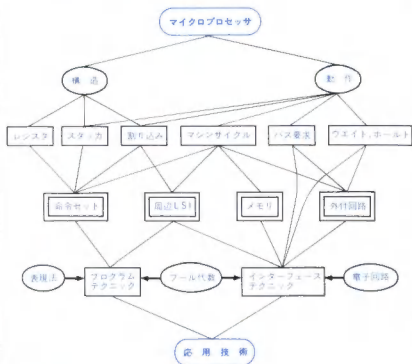
その 8 ビットマイコンの中でも Z-80 は、世界標準と呼ぶに相応しく、オリジナルソースであるザイログ社にとどまらず、国内主要メーカー各社からも Z-80 をベースにした発展型が発表され、好評を博しております。Z-80 の基本設計のよさが認められた結果、多くのエレクトロニクスエンジニアが Z-80 を必要条件と考え習得するに至ったため、以降の 8 ビットマイコンはこれに準拠することが市場に受け入れられる最短距離と、だれしもが認めるところとなったのです。

新たな展開の時期に至り、Z-80を踏み台にして飛び立って行ったマイクロコンピュータ自身の軌跡を、これからマイクロコンピュータを学ぼうと志す若いエンジニアの指針とするべく、本書の内容も、より適切な形に改めることにいたしました。次の世代のエレクトロニクスエンジニアの必携ツールである、マイクロコンピュータ応用技術の第一歩を本書から読み取っていただければ望外の幸いです。

1993年7月

著 者 し る す

## マイクロプロセッサの応用技術



## 基礎知識

- |                                       |                             |                                       |
|---------------------------------------|-----------------------------|---------------------------------------|
| ◦ 表現法<br>(用語)<br>(16進表現)<br>(アセンブリ言語) | ◦ ブール代数<br>(真理値表)<br>(論理回路) | ◦ 電子回路<br>(オームの法則)<br>(電子部品)<br>(半導体) |
|---------------------------------------|-----------------------------|---------------------------------------|

## 付加事項

- 経験と知識に基づく 慣れやマカン を養うこと。  
自分でやってみることが重要
- 中身がどうなっているかは問題ではない。  
どうすればどうなるかを知って使いこなすことが重要。

---

## 学習のポイント

---

マイクロコンピュータは、ハード面、ソフト面が有機的に結合しあいシステムを形成しています。ページ展開に従った一次元的な説明では到底表現しきれないものです。平面的、二次元的つながりを重要視する意味で項目間の参照は不可欠です。また、一読して理解できないことを嘆くなら再読を勧めます。一回目ではほんやりと全体像をとらえ、2回目では後ろに書いてあることを思い起こしながら前の説明を見て行きます。次に自分でプログラムを考えてみて下さい。例題のプログラムを自分なりに手直しするだけでも良いのです。自分自身で試行することにより不明点をとらえ、解説を読み直せば確実に身につきます。

- (1) マイクロコンピュータは物性や自然科学とは異なり、人間が考え出して作ったものです。使い易く、覚え易くするためにさまざまな知恵を絞ってあります。マイクロコンピュータを理解するのは、作られたルールを習得することととらえ、自然科学の探求とは異なった考え方で対処すればいたって気軽に取り組めると思えます。
- (2) また、マニュアルのような正確緻密な表現よりも、平易な解説を心がけました。コンピュータを理解できないという人のほとんどが、聞き慣れない用語とアルファベットの羅列に面食らっているようで、この点も読み進むうちにだんだんとなじめるように配慮したつもりです。
- (3) ただ、マイクロコンピュータを学習するうえで、オームの法則程度の電子回路、ブール代数、論理回路(ロジック IC)等についての最低限の知識は必要です。この点は他に良書が多数あることを理由に、触れる程度に止めました。



- (4) マイクロコンピュータを学習する上で、まず『何をしたいのか』といった具体的テーマをもち、『何ができるのか』からのフィードバックにより回答を模索して行くことが、一番の近道であると思います。この作業は物作りすべてにあてはまる事であり、エンジニアリングの本質でもあると思います。



# 目 次

1	マイクロプロセッサの特質	2
2	マイコンシステムの種類	4
3	特質を活かすシステム設計	6
4	Z-80 ファミリの特徴	8
5	CPU コア ASIC	10
6	Z-80 CPU	12
7	Z-80 PIO	14
8	Z-80 CTC	16
9	Z-80 DMA	18
10	Z-80 SIO	20
11	メモリの種類と用途	22
12	ビットパターンと16進表現	24
13	プログラムの実行	26
14	CPUの信号のやりとり	28
15	データバス、アドレスバスとシステム制御信号	30
16	命令語の構成	32
17	命令の実行	34
18	アセンブラ記法のルール -1-	36
19	アセンブラ記法のルール -2-	38
20	命令の分類	40
21	メモリ空間とIO空間	42
22	アドレスデコーダ	44
23	バスバッファ	46
24	システムクロック	48
25	リセット	50

26 システム構成	52
27 フェッチサイクルの動作	54
28 メモリリードサイクル	56
29 メモリライトサイクル	58
30 IO リードサイクル, IO ライトサイクル	60
31 リフレッシュサイクル	62
32 CPU と周辺の接続 (インターフェース)	64
33 ウェイト信号とホルト信号	66
34 割り込みの概念	68
35 ノンマスカブルインタラプト ( $\overline{\text{NMI}}$ )	70
36 インタラプト ( $\overline{\text{INT}}$ )	72
37 モード0のインタラプト	74
38 モード1のインタラプト	76
39 モード2のインタラプト	78
40 デーजीチェーン	80
41 バス要求と応答	82
42 内部レジスタの構成	84
43 A, I, R, F レジスタ	86
44 汎用レジスタ	88
45 補助レジスタと交換命令	90
46 IX, IY レジスタ	92
47 スタックとスタックポインタ (SP レジスタ)	94
48 転送命令	96
49 算術演算命令	98
50 論理演算命令	100
51 ビット操作命令	102
52 ローテート, シフト命令	104
53 ブロック転送, ブロックサーチ, ブロック入出力命令	108
54 ジャンプ命令	112

55	コール、リスタート、リターン命令 (サブルーチン).....	116
56	Fレジスタとフラグ変化 .....	118
57	2進化10進数と10進補正命令 .....	120
58	ペリフェラルのプログラミング .....	122
59	PIOモード0の動作 .....	124
60	PIOモード1の動作 .....	126
61	PIOモード2の動作 .....	128
62	PIOモード3の動作 .....	130
63	PIOのプログラミング .....	132
64	PIOのプログラム例 .....	134
65	CTCカウンタモード.....	136
66	CTCタイマモード.....	138
67	CTCのプログラミング.....	140
例 題		
	プログラム 1 (ループ).....	144
	プログラム 2 (判断).....	146
	プログラム 3 (メモリクリアサブルーチン).....	148
	プログラム 4 (変換テーブルサーチ) .....	150
	プログラム 5 (スイッチの表示).....	152
	プログラム 6 (スイッチの表示-割込み) .....	154
付 録		
	付1 Z-80 命令表 .....	157
	付2 Z-80 規格表 (参考) .....	176
	索 引 .....	189





Z-80 学習用に最適なポケットコンピュータ PC-E200(シャープ)

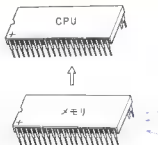
## マイクロプロセッサの特質

コンピュータという機械が発明された目的は、人間には不可能な複雑な計算を短時間に行うことでした。たとえば円周率の計算を一生の大半を費やして行っても数百桁だったのに。コンピュータなら瞬時にもっと多くの桁を間違いなくやってのけることができます。人間を月へ送り込むのも、正確な軌道計算が刻々と得られるためにできたことで、コンピュータなしでは考えられないのです。ある手順の決まった仕事を、高速かつ正確に処理することのできる機械、これがコンピュータの目的であり、その目的は十分に達せられたといえるでしょう。

ところが、ストアプログラム方式すなわち、処理手順を示すプログラムを、コンピュータに入力すべき情報の一つとして扱うことのできる今日のコンピュータ方式では、当初の目的以外に、大きな特徴が認識されました。全く同一の機械（ハードウェア）に対し、使用者が作成するプログラム（ソフトウェア；これは使用者によってそれぞれに異なる）をメモリに入れることにより、使用者のそれぞれの目的に合った仕事をする、という特徴です。コンピュータメーカーは同じ機械を大量に生産すれば、科学計算であろうと、在庫管理であろうと、人事管理、生産ラインコントロール、座席予約、相性判断まで、プログラム次第で適用されてしまうのです。

半導体の技術が進歩して数千、数万の部品を数ミリ四方の中へ作り込めるようになったとき、専用の機能を持ったLSIが、さまざまな目的に合わせて作られました。ところが大量生産にしか向かないLSIの弱点を補うべく、半導体メーカーは汎用LSIの思想をコンピュータに見出したのです。コンピュータをLSI化すれば、多く用途のある、つまり大量に売れるLSIが作れると考えたのです。マイクロプロセッサはこのような中から誕生し、プログラムによって機能が決定される汎用の論理素子として、すばらしい発展をとげ、現在も発展しつつある、“電子部品”なのです。マイクロプロセッサを中心としたマイコンシステムを利用しようとする場合、この辺を的確に把握してかかることが重要です。





→ 同じLSI でも

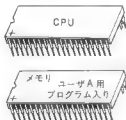
→ 同じロジックボード  
(電気回路)でも

ユーザA ゲームのプログラム

ユーザB 自動販売機のプログラム

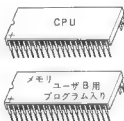
ユーザC 数値制御のプログラム

メモリに入れるプログラム次第で



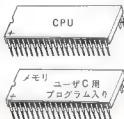
→ ユーザA 用コントローラ

ゲームマシン



→ ユーザB 用コントローラ

自動販売機



→ ユーザC 用コントローラ

NC 工作機械

## マイコンシステムの種類

**マ**イコンと呼ばれるコンピュータシステムには、いくつか考えられます。一つに、パーソナルコンピュータやオフィスコンピュータに近い性能を持つものを“マイコン”と呼びます。これらは、コンピュータを小さく、安価にまとめ、事務所や大学の研究室や個人の知的玩具としてまで普及した個人用コンピュータシステムです。電気の知識がなくても、簡単な言葉を覚えれば誰でも使えるテレビ、オーディオなどと肩を並べる全く新しいマニアライクな道具といえます。この場合のマイコンは、“My Computer”と解釈すべきでしょう。

もう一つの“マイコン”は、パソコンほどのはなばなしはありませんが、工作機械や自動販売機や家電製品に組み込まれている、機器制御用の**ワンボードマイコン**です。基板上に作られたマイクロコンピュータシステムは、産業用としての中心をなすものです。各種の用途でそれぞれの目的に合った設計がなされ、特徴を出しています。組み込まれた時点では、プログラムは固定化され、機器のユーザには、コンピュータとしての使用はできないのが普通です。

1個のLSI上にメモリや入出力ポートを作り込んだ**ワンチップマイコン**は、時計や電卓に使われています。中身を分析すれば、マイクロコンピュータに違いありませんが、LSIを外面から見た場合、一つの機能を持った専用LSIになってしまいます。開発過程では、コンピュータの特徴が活きて、短時間に安価な開発経費で専用LSIが作れるのです。機器制御用マイコンシステムをワンチップ化したと考えればよく、生産数量がきわめて大きい場合、コストやサイズ、消費電力などの点で、効果をあげることができます。

また、マイコンシステムを構成する最重要部品であるマイクロプロセッサ、つまりCPUのLSIを“マイコン”と呼びます。しかし、これだけではコンピュータとはいえず、周辺やプログラムを含めてコンピュータシステムが完成するのですから、あくまでCPUと呼ぶべきです。



マイクロプロセッサ  
(CPU)



入出力LSI



カウンタタイマ  
など



メモリ



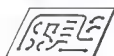
ロジックIC



抵抗



コンデンサなど



基板  
コネクタ  
など



電源



ブラウン管  
プリンタ  
など



キーボード



フロッピー装置  
など

ワンボード  
マイコン  
(制御用)

ソフトウェア



パーソナルコンピュータ



## 特質を活かすシステム設計

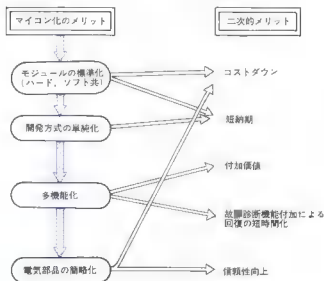
コンピュータは、仕事を高速に処理することと、同じ機械(ハードウェア)でもプログラム(ソフトウェア)を変えるだけで異なった働きをさせられること、の2点が最大の特徴です。制御用マイコンは、メカニズムやリレーや論理回路によるワイアードロジックからの延長線上にあり、高速性では、ワイアードロジックには一歩ゆずります。むしろ、プログラムにかかる機能面の柔軟性が大きなメリットです。また、多機能化することが比較的容易であり、故障時に自己診断をさせることすら可能です。マイコンボードとして決定された能力の限界以内であれば、同一の規格化された基板を何種類もの製品に应用することができ、開発コストの引き下げに効果が期待できるようになります。プログラムもモジュール化し保管をすることにより、経験の蓄積ができ開発期間の短縮が図れます。

マイコンシステムの設計にあたっては、ハードウェア(以下、ハードといいます)とソフトウェア(以下、ソフトといいます)の分担を決定することがポイントになります。回路設計とプログラム設計の前段階としてのシステム設計といわれる作業です。往々にして、電子技術者が設計したシステムはハードにウエイトがおかれてしまう傾向があります。ハード、ソフト共に精通した人が、目的の機能と、開発コスト、ランニングコスト、メンテナンスコストなどすべてにわたって検討して決定すべきです。特に部品やボード、プログラムの標準化という意味から、将来にわたって要求される機能の変動を見通すことが重要になります。変動要素はソフトの分担とし、メモリの交換だけで対応できなければなりません。

概してコンピュータの特質を考えるならば、タイミング的に不可欠な部分をハード化し、他はソフトで実現する方法に利があるといえます。ハードを最小限に止めることは、材料費、経年劣化、故障率を低減するうえで有利であることは明白です。

## マイコン適用分野

- 多機能であって、ICレベルの論理素子で構成すると、大規模になりすぎるもの  
——→ パーソナルコンピュータ
- 大同小異の機種が多く、いちいち論理素子で構成したのでは設計に手間がかかりすぎるもの ——→ 端末装置、NC 機器
- 納入先ごとの変更があるもの ——→ ビル防災システム、ホテル管理システム
- あとで変更のあり得るもの ——→ 自動販売機、料金計算機
- 開発時間の限られる場合 ——→ ハードウェアは標準品を使い、プログラムだけを入れ換える



## 必要条件

- 改善意欲 ——→ 社内規格の統一・コストダウン
- 先行投資 ——→ 開発装置購入
- 技術力 (技術吸収力) ——→ 要員教育

## Z-80 ファミリの特徴

**Z**-80 はインテル社が開発した 8 ビットマイコン 8008 からの進化の流れをくみ、8 ビットプロセッサとして一応の完成を見た製品です。デビューのタイミングがマイコンの普及期と一致したこともあり、多くの技術者に受け入れられた、標準的な体系を備えた汎用プロセッサです。

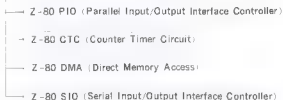
5 ボルト単一電源、単相クロック、16 ビットの減算などの機能を持ち、発表当時としては極めて先進的なマイコンでした。さらに機能を充実した 8 ビットマイコンは Z-80 以降も開発されています。しかし基本的な考え方は、Z-80 を踏襲したものがほとんどです。すでに普及した基本体系を変えないことが、開発装置などの環境や技術者のノウハウをはじめとする蓄積資産の有効活用を可能にし、容易に市場展開ができるのです。

初期の Z-80 ファミリは、N-MOS プロセスでしたが、その後より低消費電力を求めて C-MOS プロセスへ移行し、処理速度、動作温度範囲、動作電圧範囲、パッケージなどユーザの選択範囲を広げるべく、いろいろなバリエーションがそろっています。

機器の小型化の動きは複数の LSI を一つの LSI に集積することさえ要求し、コア方式といった単純化・画一化された開発手順で、用途別に特化された IC「ASIC」を次々に生み出すことが可能になりました。Z-80 CPU を核とし周辺機能を取り入れたワンチップマイコンも各社から発表され、プログラム領域を外部におくことのできるタイプのものは、量産数の少ない用途でも高集積 LSI のメリットを取り入れることができ、喜ばれています。組み込む周辺機能はいろいろな組み合わせがあり得。メーカーでは各種の製品をそろえています。最近では、Z-80 の弱点とされていた、乗除算命令などを追加した CPU コアもあり、また Z-80 ファミリ以外の A/D コンバータやオペアンプ、コンパレータなどもそろえて、ますます選択の幅が広がり、使いやすくなっています。

## Z-80 ファミリ

## Z-80 CPU (Central Processing Unit)



## 特 徴

- 8ビット標準アーキテクチャ
- 割り込み機能 → コントローラ不要、プログラムが簡単
- レジスタ群 → プログラム容量の圧縮・スピードアップ
- 命令セット → プログラム容量の圧縮・スピードアップ
- リフレッシュ機能 → ダイナミックメモリ使用可能 → コストダウン

## 命令セットの特徴

- 8ビットマイコン標準的命令体系
- レジスタ、メモリのビットのセット、リセットテスト
- メモリブロックの転送、サーチ、入出力
- 2の補数をとる命令
- 4ビット単位のローテーション
- 整理された二モニック（暗記用命令コード）

## CPU コア ASIC

マイコンは大量生産の得意な半導体メーカーと、少量生産でも大規模集積回路の恩恵を蒙りたい電子機器メーカーのギャップを埋める、虹のかけ橋として登場したわけです。Z-80 ファミリでは個別の機能ユニットをそれぞれの LSI としてまとめ、ユーザは必要とするシステムを、LSI の組合わせで構築するという方法で、これを実現してきたのです。ところが、小ささへの価値観が高まるにつれて、複数 LSI の組合わせを 1 個の LSI に集積したいという要求が強くなり、比較的量産数の少ない分野でもワンチップマイコン導入の動きが起き、しかも応用分野ごとの異なった要求仕様を満足する必要が出てきました。半導体メーカーは、画一的な製品構成から抜け出て、応用範囲を拡大すべく、それぞれのユーザの要求に合う製品をラインナップしたいところですが、これまでの大規模集積回路の開発手法では自ずと限界があり、対応を模索してきました。

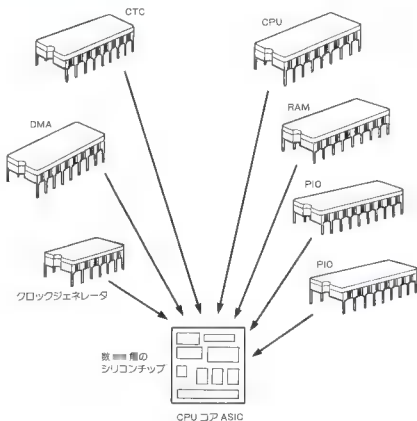
ASIC (Application Specific IC) [エイシック] というのは、特定用途向け IC と訳され、適応範囲を広げることをご考慮せず、文字どおり特定用途だけを対象に考えて、LSI を開発する手法あるいはその製品のことで、新たな要望仕様に対するカスタム LSI を比較的容易に実現できるものです。個別の LSI に相当する機能セルを、それぞれの応用に最適な組合わせで選択し、一つの LSI 上に作り込む方法で、システムオンチップとも呼ばれます。

CPU コア ASIC は CPU を核として必要な機能を組込んだ LSI で、あたかもプリント基板上に CPU と周辺 LSI を並べてできるワンボードコンピュータのように、シリコンチップ上に機能セルを並べてできるワンチップマイコンなのです。この CPU コアとして、最も普及している Z-80 CPU を使用できるメリットははかり知れず、すでに数社の半導体メーカーから発表されています。

しかし、いかに ASIC といえどもある程度の数量規模がないと生産ラインに乗せることはできませんし、数量規模が見込めれば価格的に有利になります。半導体メーカーでは数量規模がないユーザのために、いろいろな組合わせを持ったコアマイコンをあらかじめ用意し、対応しています。



## システム構成に必要なLSIの組合せ



必要な機能を一つのシリコンチップ上に集積したIC



用途別ワンチップマイコン



省スペース、省電力、信頼性向上

## Z-80 CPU

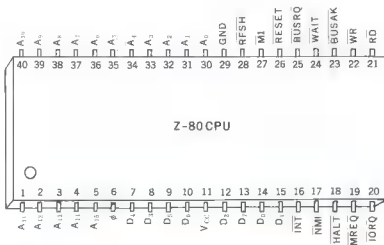
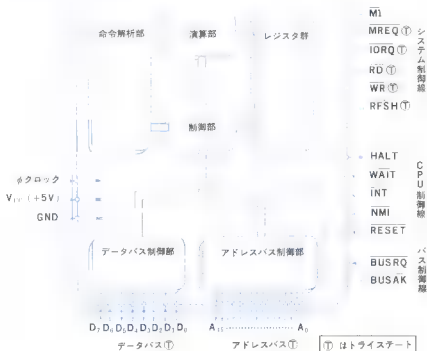
**C**PU (Central Processing Unit) は Z-80 ファミリの中核をなすものであり、ファミリの性格を決定する中心要素です。なお、ファミリで CPU 以外の LSI を、周辺 LSI とかベリフェラルと呼ぶことがあります。

Z-80 CPU には 8 ビットの汎用レジスタが 12 個あります。2 個をつなげて 16 ビットレジスタとして使用することもできます。インデックスレジスタは 2 個あり、他にスタックポインタ、プログラムカウンタ、割り込みベクトルレジスタ、メモリリフレッシュレジスタ、フラグレジスタ、アキュムレータがあります。

これらのレジスタや命令を解析し実行する制御部や演算部の働きは、以下の項で解説します。

外部端子には、アドレスバス 16 本、データバス 8 本、システム制御線 6 本、CPU 制御線 5 本、バス制御線 2 本、それにクロックと電源が出ています。図で、トライステートと記入されている端子は、“H”と“L”の二値状態のほかに、端子と内部が電氣的に切り離された“ハイインピーダンス”状態を持つものです。動作と関係のないときはこの状態になっており、外部からの負荷にならないようになっています。

Z-80 ファミリは、使用可能なクロック周波数により、いくつかのバージョンに分けられています。また、停止時に消費電流を減らす機能が付いたものや、特に高信頼度の要求される用途向けなど、さまざまなバリエーションのデバイスが作られています。



## Z-80 PIO

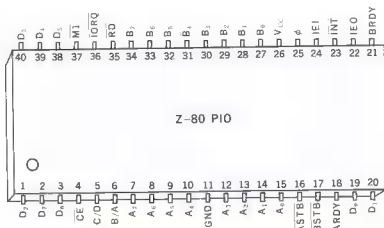
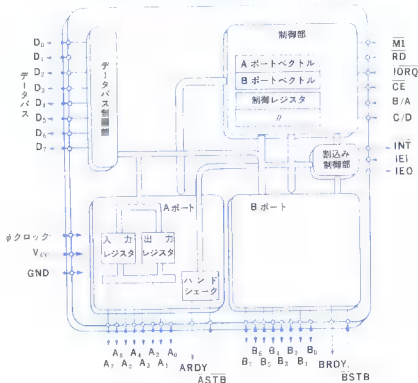
**C**PU の次によく使われるのが、この **PIO** (Parallel Input/Output Interface Controller) です。CPU からの信号を受けて、外部の装置に出力したり、外部装置からの信号を受けて、CPU へ伝えたりするデバイスです。

CPU の信号は、データバスを入出力以外の用途にも時分割で共用しているため、大変複雑な信号になっています。PIO は、この中からある特定の時点のデータバス上の信号をとらえて外部との受け渡しをするためのもので、一般にラッチと呼ばれるロジックに、使いやすい機能を付け加えた LSI といえます。信号は 8 ビットを並列に扱います。動作は次の四つのモードがあります。

モード 0	出力モード	} ハンドシェイクにより 8 ビットデータを 入・出力する。
モード 1	入力モード	
モード 2	入出力モード	
モード 3	ビットモード	(ビット単位に入力、出力を選択できる)

モードやその他の動作条件は、CPU からの信号により内部の制御レジスタに制御ワードを書くことによって設定されます。割り込みは CPU と PIO (または他のペリフェラル) のみで制御され、優先順位の決定や、どのペリフェラルからの割り込みかの解析や、割り込み処理プログラムルーチンからのリターンは、自動的に行われます。

PIO の内部にはほぼ同一のポートが二つあり、それぞれ別の動作モードで使うこともでき、割り込みも 2 系統発生させることができます。優先順位は、A ポート → B ポートの順です。



## Z-80 CTC

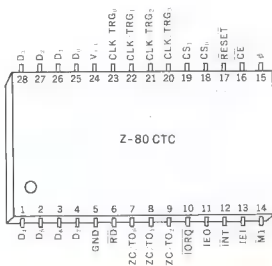
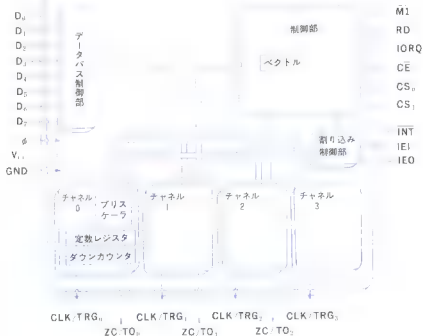
**C**TC (Counter Timer Circuit) は、パルスのカウントダウンをする LSI です。不特定周期の外部パルスをカウントし、設定数になると割り込みを入れたり、ゼロカウント出力を出したりすることができます。不特定なパルスでなくクロックのような決まった周期のパルスをカウントすることにより、タイマとしても使えるわけです。

何個目のパルスで割り込みまたはゼロカウントするかの設定や、カウントパルスのエッジ（立ち上りか立ち下りか）の選択などは CPU からの書き込みによってプログラムされます。またカウントの途中での残り数は、CPU がカウンタの内容を読み出すことによって知ることができます。

設定できる数は 1~256 までですが、くり返して何回日かの割り込みでカウント終了することにより範囲は広がります。また、ゼロカウント出力を次チャンネルのカウンタへ入れてやれば、 $256^2$  までが扱えます。したがって、 $n$  個つなげば  $256^n$  となります。

1 個の CTC には四つのチャンネルが内蔵されていて、別々の目的に使用することができます。ただしピン数の関係で 4 個目のチャンネルは、ゼロカウント/タイムアウトの出力が出ていませんので、割り込みによりカウント終了となる使い方しかできません。

システムクロックのカウントダウンによるタイマとしての使用時は、プリスケールによってクロックを 16 または 256 分周したパルスをカウントします。タイマの起動は自動的に行うことも、また外部トリガによって行うこともできます。



## Z-80 DMA

× モリ内、あるいはメモリと入出力 (IO) ポートとのデータの転送は、通常は CPU が一度レジスタへ読み込み、次に書き出すという方式で行います。Z-80 には、単命令で一連の複数のデータを転送するブロック転送命令があります。プログラムステップ数は少なくなり、便利ではありますが、転送に要する時間は、1 バイトずつくり返しループを実行して、転送するのと大差ありません。この転送の作業だけを高速に行う周辺 LSI が **DMA** (Direct Memory Access) です。

CPU から、あらかじめ転送されるデータの入った元のアドレス (ソース) と転送先のアドレス (ディスティネーション) と、転送するバイト数を DMA 内のレジスタへ書き込めば、自動的に指定の転送を行います。ソースまたはディスティネーションは、自動的にカウントアップされますが、特定の入出力ポートへ次々に出力または入力するときはカウントアップを止めておくこともできます。

転送だけでなく、転送しながら、あるいは転送をせずに指定されたビットパターンとデータのビットパターンの一致をチェックすることもできます。これを**サーチ**と呼んでいます。サーチの終了 (一致) は、転送終了とは区別できるよう割り込みがかかります。

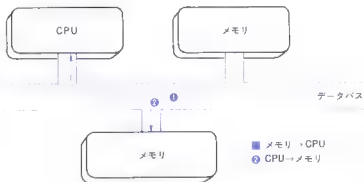
DMA が動作中は、アドレスバスやデータバスは占有されますので CPU は待ち状態になりますが、指定により CPU 優先として 1 バイト単位に転送することや、外部ロジックからの切り換えにより CPU へバスを明け渡すこともできます。

DMA や SIO を使うシステムは、かなり大がかりなものになります。

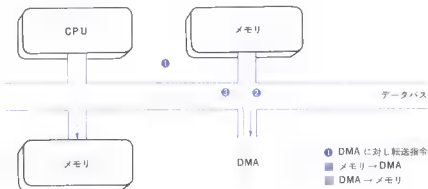
本書では、初歩的な内容をわかりやすく解説することが主目的ですから、これらを対象からはずし、次のステップでの修得を期待します。



## 通常の転送（メモリ→メモリの場合）



## DMA 転送（メモリ→メモリの場合）



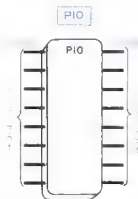
## Z-80 SIO

**P** IO は、データを並列に入出力するためのポートコントローラですが、この **SIO** (Serial Input/Output Controller) は、データを直列に、すなわち時間経過に従って入出力するポートコントローラです。電話回線を使って長距離の通信を行なう場合はもとより、同一機器の筐体内でもケーブルが長くなる所は、データ通信線の数が少なくすむシリアル伝送を利用することがあります。もちろん他の条件が同一なら伝送速度はパラレルの 1/8 になることはやむを得ません。

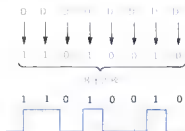
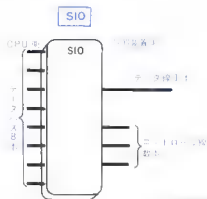
シリアル伝送の方式には、同期・非同期、バイト指向・ビット指向などがあり、また通信速度や電圧、電流など、何種かの規格があります。コントロールプログラムと外部付加回路によってほとんどの方式に対応できる、きわめてぜいたくな機能を持った LSI です。

この SIO のチップ (LSI 中のシリコンウェハ上の本体) は 41 個の外部端子を持っていますが、パッケージは 40 ピンであるため、外部ピンに接続されない端子があるもの 2 種類と、2 本の端子を 1 本のピンに接続したもの 3 種類の SIO があります。これをボンディングオプションと呼んでおり、SIO-0、SIO-1、SIO-2 と区別しています。

SIO はきわめて多機能であるため、すべての動作を理解しようとせず、シリアル伝送の方式を理解したうえで、SIO の機能の必要な部分を使用するよう考えるのが効果的です。



- 外部装置から送り込まれるデータバスの内容を、ある瞬間をとらえて (IO サイクルに) CPU 側のシステムデータバスに送り出す。
- CPU 側のシステムデータバスの内容を、ある瞬間をとらえて保持し、外部装置への受け渡しのタイミングをとる。



- 外部装置から送られてくるデータ線上のパルス列を、8 ビットの並列データになおし、CPU 側へ送り出す。
- 8 ビットの CPU 側データバスの内容を、時間と共に順次、データ線へ送り出す。
- 上記に必要なコントロール信号、内容の検査をするための付加情報を扱う。

## メモリの種類と用途

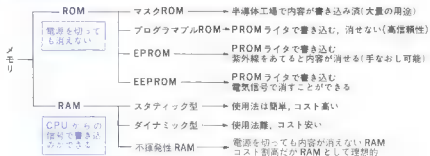
**マ** イコンシステムで使われるメモリには何種類ありますが、現在ではほとんどが半導体素子で構成される LSI メモリです。

LSI メモリは **ROM** (リードオンリーメモリ) と **RAM** (ランダムアクセスメモリ) に大別されます。ROM は書き込みに特別な装置が必要で、CPU からの書き込みはできません。ただし電源を切っても内容は消えないため、プログラムを入れておくのに主に使われます。RAM は、CPU からの書き込みができますが、電源を切ると内容は消えてしまいます。

ROM には、**マスク ROM** と **PROM** (プログラマブル ROM) があります。マスク ROM は半導体工場で生産する時点で内容が決定されてしまいますので、同一内容の量産に適しています。PROM は、ROM ライタという装置で書き込みますが、一度書いたら変更できないもの (**バイポーラ型 PROM**) と、電気信号で消去できる **EEPROM** (エレクトリカルイレーザブル PROM) と紫外線照射により消去できる **UVEPROM** (ウルトラバイオレットイレーザブル PROM) があります。一般的には UVEPROM を単に **EPROM** と呼び、多く使用されています。

RAM は **SRAM** (スタティック RAM) と **DRAM** (ダイナミック RAM) があります。SRAM はフリップフロップにより構成されたメモリで、使いやすい特徴があります。DRAM はコンデンサの電荷蓄積を応用したメモリですから、時間と共に消えてしまいます。消えないうちに (実際には 2ms 以下ごと) に一度読み出して、再び同一内容を書き込むようにします。これを **リフレッシュ** といい、Z-80 では、CPU が命令コードを解析している時間を利用して自動的にリフレッシュする機能を持っています。コスト的にはメリットがあるのですが、ドライブのインタフェースが複雑になり、むずかしさが残ります。最近ではメモリ内部にリフレッシュ機能を持ち、疑似的に SRAM と同様に使える DRAM もあり、**疑似 SRAM** と呼ばれています。また、電源を切っても内容の消えない不揮発性 RAM、**NVRAM** (エヌバイラム) が普及しはじめ、理想的な半導体メモリとして、多方面に使われるようになってきています。

## 使い方による分類



主に

ROMはプログラムを書いておく

RAMは一時記憶用データ格納

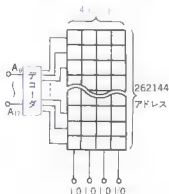
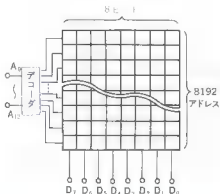
読み出しだけだが電源を切っても消えないから

読み書きできるから

## よく使われるメモリの例

EPROM (2764)

SRAM 1M (256K × 4)



メモリは容量、生産工程、パッケージなどが各種あります。メーカーのマニュアルによって適当なものをさがすことも重要な仕事です。

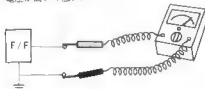
## ビットパターンと16進表現

**C** PU 内部や入出力信号線は、電氣的に電圧が高い状態と低い状態の二つで、意味を持つようになっています。1本の電線では信号が有る、無い、の二つの状態でしかあり得ませんが、複数の電線を一束にして考えると、高・低の組み合わせは2の累乗で増加します。アドレスバスは16本ありますので、16本の電線の高・低の組み合わせは $2^{16}=65536$ とおりになります。すなわち、CPUの出すアドレス情報は65536のメモリを識別できることになります。データバスは8本ですから、データとしては $2^8=256$ とおりの種類が得られます。しかし数回に分割してやりとりをすれば、理論的には無限の組み合わせが得られます。8本では数値としては0~255までしか表現できないはずですが、実際にはもっと広い範囲の数値を扱えるのはこのためです。

一つの2値状態を表現する単位を**ビット**と呼びます。アドレスバスは16ビットです。電圧の高い状態のことを“H”または“1”と呼び、低い状態のことを“L”または“0”と呼びます。16ビットの状態を表現するのに“1”と“0”を16個並べてもよいのですが、長くて扱いにくいので困ります。そこで“1”と“0”の組み合わせ(**ビットパターン**)を2進数とみなして、これを16進数に変換して扱うと大変便利になります。要するに、ビットパターンを4桁ずつ区切って“1”、“0”の組み合わせに名前を付けたと考えればわかりやすいでしょう。名前は数字の0~9、次が10でもよいのですが、2桁になってしまうのでAと呼び、次にB、C~Fまであります。0~Fで16ありますので16進数になるのです。本書でもビットパターンをいちいち書くのは大変なので、16進数で表現します。その場合は後にHを付けて10進数と区別します。Z-80のアセンブリ語でもこのように書く決まりになっています。またマシン語の命令も本来はビットパターンで表現されるはずですが、一般的に16進数表現で扱います。

演算はビットパターンを2進数として行います。普通は0~255(8ビットでは)になりますが、符号を付けて-128~+127として考えることもできます。どちらをとるかでプログラムは異なります。

- ① 電圧が高い(低い)



- あながあいている(あいてない)



- スイッチが上に倒れている(下に倒れている)



- ランプが点灯している(消えている)



などを“1”というとき、( )内のときは“0”という→ 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1

※ 4つ要素が並ぶとすると下のとおり16とありのパターン(組み合わせ)が考えられる。

0 0 0 0 0  
0 0 0 1 1  
0 0 1 0 2  
0 0 1 1 3  
0 1 0 0 4  
0 1 0 1 5  
0 1 1 0 6  
0 1 1 1 7  
1 0 0 0 8  
1 0 0 1 9  
1 0 1 0 A  
1 0 1 1 B  
1 1 0 0 C  
1 1 0 1 D  
1 1 1 0 E  
1 1 1 1 F

いちいち書くのは大変なのでパターンに名前をつけることにした(右欄)。

要素が4つだから →  $2^4 = 16$  とおり

要素が8つだったら →  $2^8 = 256$  とおり

要素が16だったら →  $2^{16} = 65536$  とおり

※ パターンを2進数と見なすと名前は16進数になる。  
コンピュータ内の演算はこのパターンを2進数と見なすことにより数値を表現して行われるようになっている。

※ たった8本の電線(信号)でも256とありの状態を表現することができる。

- “1”のことを“H”(high)、“0”のことを“L”(low)と呼ぶこともある。
- 普通は“0”になっていて、“1”になったら「信号がある」と決めておけば、“1”のことをアクティブという。
- 普通は“1”になっていて、“0”になったら「信号がある」と決めておけば、“0”のことをアクティブといい「この信号は重論理だ」という。

## プログラムの実行

× モリに書き込まれているプログラムは、マシン語です。マシン語は英数字の組み合わせで、人間の目にはきわめてわかりにくいものですが、たとえば「A と B を加えて A に入れよ」とか、「止まれ」とか、「A の内容を 10 番のポートへ出力せよ」といった命令にそれぞれ付けられた番号なのです。CPU は、日本語や英語で書かれるより、番号のほうが簡単に見分けることができます。

プログラムの書き込まれたメモリには、これも番号＝番地（アドレス）が付けられています。実行するときは、まずゼロ番地に書かれた命令から順次 1 ずつくりあげていきます。ところが命令の中に「何番地へ飛び」というのがあったら、1 ずつくりあげるのではなく、指定の番地へ飛び（ジャンプ）ます。また「前の演算の結果がゼロならば何番地へ飛び」という命令では、条件によって、ゼロなら飛び、ゼロでなければ次の番地の命令を実行します。

サブルーチンコール命令があると、指定された番地へ飛び、そこから順次実行し、最後に付けられたリターン命令で、先ほど飛んできた番地の次へ戻ることができます。同じ手順を何度も使いたいときによく使う方法で、飛び前のプログラムをメインルーチン、飛んでくるプログラムをサブルーチンと呼びます。

サブルーチンとよく似ているのが割り込みです。割り込みは、命令があつて特定の番地へ飛び、のではなく、電気信号が CPU の割り込み信号端子に与えられると、どこの命令を実行中であってもある特定の番地へ飛び、リターン命令で元のメインルーチンへ戻ります。この特定の番地に置かれたプログラムを割り込み処理ルーチンと呼ばれます。

CPU 内には、プログラムカウンタ (PC) と呼ばれるレジスタがあり、この内容で、いま実行すべき命令の入っているメモリのアドレスを示すようになっています。一つの命令を実行するたびにこの PC を増やし、次の命令のアドレスを指します。また、特定のアドレスへジャンプするようなときは、PC へ飛び先のアドレスを強制的に入れることにより目的を達成します。



Aの内容をBへ入れよ	= 47
止まれ	= 76
AにBを加えよ	= 80
200番地の命令へ行け	= 03 00 02

0000	47
1	80
2	03
3	00
4	02
.....	.....
0200	47
1	29
2	76
.....	.....

(注) ニコではビットパターン  
の名前で書いたが、メモリに  
はビットパターンそのものが  
記憶されている。

- メモリにはプログラムを命令番号で入れておく。
- 最初は0000番地の命令を実行する。
- 次々に先の番地を順序よく実行する。
- 0002番地にある命令(200番地へ行け)を実行すると、次は200番地にある命令を実行する。
- 次々に先の番地を順序よく実行する。
- 0202番地にある命令(止まれ)を実行すると、そこで以後の実行をやめ停止する。

## CPU の信号のやりとり

**C**PUにはたくさんの信号入出力線が出ていますが、これらの信号線を使って、基本的な六つの動作を時分割的に行います。それぞれを**マシンサイクル**と呼びます。

### 1. フェッチサイクル (M1 サイクル)

メモリに書き込まれているプログラム命令を CPU 内の命令解析用レジスタ (インストラクションレジスタ) へ読み込み、解読する。

### 2. メモリリードサイクル

メモリからデータを CPU 内のレジスタへ読み込む。

### 3. メモリライトサイクル

CPU 内のレジスタからデータをメモリへ書き出す。

### 4. IO リードサイクル

入力ポートからデータを CPU 内のレジスタへ読み込む。

### 5. IO ライトサイクル

CPU 内のレジスタからデータを出力ポートへ書き出す。

### 6. リフレッシュサイクル

DRAM をリフレッシュするためのアドレス信号を出す。

以上のほかに、一連のサイクル動作ではありませんが、単独の意味を持つ信号が七つあります。

1. **ウエイト**；クロックサイクル数を増やす。

2. **インタラプト**；割り込み要求。

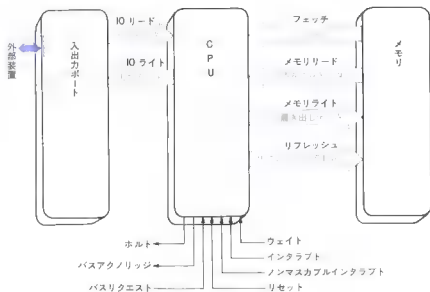
3. **ノンマスカブルインタラプト**；ノンマスカブル割り込み要求。

4. **リセット**；初期状態に戻す。

5. **バスリクエスト**；CPU の動作を停止させバスを空け渡す要求。

6. **バスアクノリッジ**；バスリクエストを受け付けた返事。

7. **ホルト**；CPU がホルト命令を実行し、停止状態に入ったことを外部へ知らせる。



は一連の動作（マシンサイクル）で、アドレス情報とデータの受け渡しを伴う。

→ は情報の受け渡しは伴わない。

## データバス、アドレスバスとシステム制御信号

**C**PUにある8本のデータバスは、CPUと外部とのデータを出し入れするための信号線です。CPUの動作サイクルによって乗ってくるデータの意味は異なります。フェッチサイクルでは、メモリから命令コードが乗ってきますし、メモリアクセスサイクルではメモリからデータが乗ってきます。またメモリアクセスサイクルとIOアクセスサイクルでは、CPU内のレジスタからのデータがデータバス上に乗せられます。

このようにデータバスは、その時々によって種々の信号の出し入れに使われます。

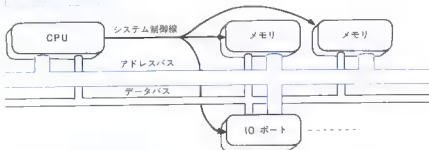
また、アドレスバスもフェッチサイクルでは実行すべき命令の入っているアドレスが出力され、メモリアクセスサイクルでは、読んだり書いたりするメモリのアドレスが出力され、また、IOアクセスサイクルでは、入出力すべきIOポートのアドレスが乗せられてきます。

データバスにせよ、アドレスバスにせよ、目的の異なる信号が次々に乗せられますので、現在の信号は何を意味するかを識別するための、別の信号線が必要になります。リード( $\overline{RD}$ )、ライト( $\overline{WR}$ )、メモリアクセス( $\overline{MREQ}$ )、IOアクセス( $\overline{IORQ}$ )、エムワン( $\overline{MI}$ )、リフレッシュ( $\overline{RFSH}$ )の各信号の組み合わせが、この役割をしています。実際にはこれらの各信号がすべて同時に出力したり、止まったりするのではなく、それぞれのタイミングで変化します。アドレスバス、データバスの占有時間も目的により異なります。

CPUの信号のうちデータバスとアドレスバスだけは正論理、すなわち、1のとき“H”レベル、0のとき“L”レベルとなります。他の信号線はすべて負論理で、普段信号のないとき“H”になっていて、必要なときだけ“L”になります。

負論理の信号名の略称には——(バー)を付けて表わします。

図 15-1-1 内部の接続



## システム制御信号線

エムワン ( $\overline{M1}$ )  
 メモリリクエスト ( $\overline{MREQ}$ )  
 IO リクエスト ( $\overline{IORQ}$ )  
 リード ( $\overline{RD}$ )  
 ライト ( $\overline{WR}$ )  
 リフレッシュ ( $\overline{RFSH}$ )

これらの組合せでアドレスバスに乗っている情報、データバスに乗っている情報あるいは、CPU がデータバスにどこから情報を乗せてもらいたい方を表現している。

- ※CPU は命令の内容によって、送り出すべき情報を送り出し、要求すべき情報を受けとる。  
情報はデータバスを使う。  
送り先や要求先は、システム制御信号とアドレスバスで指定する。
- ※メモリやIO ポートなど周辺回路は、上のCPU の要求に対しある時間内に正確に応答しなければならない。これは汎用CPU の場合ユーザの利用技術にガガっている。

## 命令語の構成

マ シン語の命令は、一つの機能を持つ命令が1~4バイトで構成されます。  
1バイト命令は、オペコードだけで意味を持つ命令です。

2バイト命令は、オペコードと、オペランドに記述された数値を次の1バイトに持つものと、2バイトで一つのオペコードを意味するものがあります。

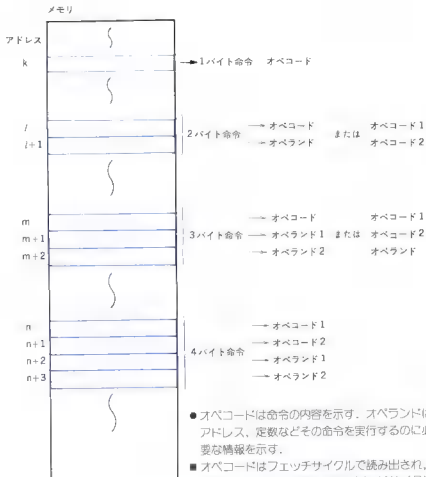
3バイト命令は、オペコードと、オペランドに記述された数値を次の2バイトに持つものと、2バイトのオペコードと1バイトのオペランドを持つものがあります。

4バイト命令は、2バイトのオペコードと2バイトのオペランド情報により構成されます。

オペコードが2バイトにわたる命令はZ-80特有のものです。最初の1バイトをフェッチし解読した時点で、2バイトのオペコードを持つことがわかりますから、あと1バイトをフェッチし解読します。したがって、フェッチサイクルが二つあり、エムワン (M1) 信号も2回出されます。

CPU は、この1~4バイトで構成される命令をフェッチサイクルとメモリーリードサイクルを起動して全部読み込み、意味を解析し実行します。これが終わると次の命令へと順次読み込み、解析、実行をくり返していきます。

一つの命令を実行する一巡を、**命令サイクル** (インストラクションサイクル) と呼びます。



- オペコードは命令の内容を示す。オペランドはアドレス、定数などその命令を実行するのに必要な情報を示す。
- オペコードはフェッチサイクルで読み出され、オペランドはこれに続くメモリリードサイクルで読み出される。

## 命令の実行

**例** として  $[LD\ A, (l/m)]$  という命令を分解してみましょう。

この命令には、3A という番号が付いていて、CPU 内部では、この番号で意味がわかるようにできています。3A の次に二つの情報、 $l$  と  $m$  が連続している 3 バイト構成の命令です。具体的には「 $l/m$  番地のメモリの内容を CPU 内の A レジスタへ入れよ」という意味があります。 $l$  と  $m$  は各 8 ビットで、メモリアドレスは 16 ビットで表わされますので、二つで一つのアドレス情報になります。このときメモリ上には  $l$  (上位) と  $m$  (下位) を反対にして、 $m, l$  の順で並べる規定になっています。

### M1 サイクル (フェッチサイクル)

プログラムカウンタ (PC) の内容で示される番地から 3A をフェッチする。CPU 内のインストラクションレジスタへ入れてこの意味を解析すると、次に何をすべきかわかり以下を実行する。PC に 1 を加える。

### M2 サイクル (メモリリードサイクル)

PC の内容番地より  $m$  を読み込んで制御部のレジスタへ一時格納する。PC に 1 を加える。

### M3 サイクル (メモリリードサイクル)

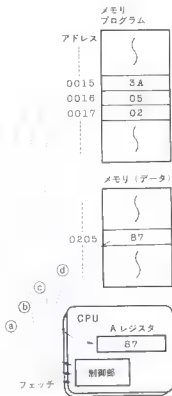
PC の内容番地より  $l$  を読み込んで制御部のレジスタへ一時格納する。PC に 1 を加える。

### M4 サイクル (メモリリードサイクル)

制御部のレジスタへ格納した  $l$  と  $m$  により一つのアドレス  $l/m$  を構成し、 $l/m$  番地の内容を読んで A レジスタへ入れる。

以上の四つのマシンサイクルによって命令を実行したわけですが、命令の内容によって、それぞれのサイクルの組み合わせは異なり、サイクル数も異なります。ただし、どんな命令でも必ず、フェッチサイクルが最初に起動され、フェッチした命令の内容によって次に何をするか決定します。





## 命令実行の例

- ① 0015 番地の命令を実行する順番がくると、CPUはアドレスバスに0015を乗せ、システム制御信号はメモリからの命令読み込みを要求する。
- ② メモリはこれに対し0015番地の中身をデータバスに乗せる。
- ③ CPUはこれを受けとり命令の意味を解析する。

## 以下解析の結果

- ④ CPUはアドレスバスに0016を乗せ、システム制御信号によりメモリからの読み込みを要求する。
- ⑤ メモリはこれに対し0016番地の中身をデータバスに乗せる。
- ⑥ CPUはこれを受けとる。
- ⑦ 同様に0017番地の中身も受けとる。
- ⑧ いま読み込んだ0016と0017番地の中身をつないで0205という値を作り、これをアドレスバスに乗せ、システム制御信号によりメモリからの読み込みを要求する。
- ⑨ メモリはこれに対し0205番地の中身をデータバスに乗せる。
- ⑩ CPUはこれを受けとりAレジスタへ入れる。

(①～⑩がフェッチサイクルで、各命令実行ごとにまずこのサイクルに入る。)

## アセンブラ記法のルール -1-

**16** 進数で表現されるマシン語命令では、人間には覚えにくくわかりにくいいため、直感に訴えるような英語の略語（ニモニック）を対応づけたのが**アセンブリ言語**です。アセンブリ言語で書かれたプログラムを**ソースプログラム**と呼びます。ソースプログラムをマシン語に、自動的に置き換えるプログラムが**アセンブラ**です。このアセンブラプログラムが読み込むソースプログラムの書き方には、一定のルールがあります。

**ラベル**はアドレスに付ける仮の名称です。この命令のアドレスへジャンプしたり、参照したりするために必要なところだけでよく、付けなくてもよいのです。英文字で始まる6文字以内の英数字列で、後で見えてわかりやすい名前を付けます。

**オペコード**は命令語のニモニックです。CPUの持つ命令の中から必要なものを選んで使います。

**オペランド**は、オペコードによってはないものもあります。一つだけのものも、“,”で区切って二つ必要なものもあります。二つ必要な命令の場合は、前に書くのが**ディスティネーション**、後に書くのを**ソース**といって、ソースからディスティネーションへのデータの移動を意味します。ソースまたはディスティネーションをカッコでくくるときは、カッコ内のアドレスのメモリに入っている内容を意味します。カッコ内がレジスタ名の場合は、そのレジスタに入っている内容をアドレスとしたメモリを意味します。

**コメント**はプログラムをあとで見るときに、わかりやすくするために必要なことをメモしておく所です。的確なコメントを豊富に付けられたプログラムは、誰が見てもよくわかり、大変便利なものです。

アセンブリ言語で書く命令には、上のようにマシン語命令に変換されるもののほかに、疑似命令といって、マシン語に変換されずに、アセンブラプログラムに対して指示を与えるだけの命令もあります。また、アセンブラプログラムが**マクロアセンブラ**と呼ばれる場合は、1命令をあらかじめ別に定義された数個以上のマシン語命令群に変換するマクロ命令といったものも使うことができます。

	ラベル	オペコード(ニモニック)	ディスティネーション	オペランド	ソース
LOOP		LD	A, B		
		ADD	A, B		
		LD	HL, 0802H		
		LD	HL, (0802H)		
		OUT	(00H), A		
		IN	A, (00H)		
		JF	LOOP		
JOB1		LD	(HL), A		
		END			

この定数が16進表現であることを明示する。

Bレジスタの内容をAレジスタへ入れよ

AレジスタへBレジスタの内容を加えよ

HLレジスタへ定数0802Hを入れよ

HLレジスタへ0802番地のメモリの内容を入れよ

00H番地のIOポートへAレジスタの内容を出力せよ

Aレジスタへ00H番地のIOポートから読み込め

LOOP番地へ飛び、その命令以下を実行せよ

HLレジスタの内容をアドレスとするメモリへ

Aレジスタの内容を書き出せ

プログラムの終了をアセンブラプログラムへ知らせる

## オペランドの表記法

	ディスティネーション	ソ ー ス
レジスタ名	レ ジ ス タ	レ ジ ス タ
定 数	—	定 数
(レジスタ名)	レジスタの内容アドレスのメモリ	レジスタの内容アドレスのメモリ
(定 数)	定数アドレスのメモリ	定数アドレスのメモリ

16進表現で、最上位がA~Fのアルファベットになるときは頭に0を付け、ラベル名と区別する。

[例] A000Hは0A000Hと書く

## アセンブラ記法のルール -2-

**行**の先頭、すなわち前の行の最後に付けられた C/R (キャリッジリターン) の次に続く文字列は、ラベルです。英大文字に続く英数字の組み合わせで、6 文字以上は無視されます。ラベルの文字列の直後に“:”コロンがあれば、行頭でなくてもラベルとみなされます。ラベルを付けない行は、1 個以上のスペースをラベル代わりに入れておきます。

ラベルと 1 個以上のスペースで区切られた後にオペコードを書きます。命令表の中から選んでください。全部で 74 あります。

オペコードと 1 個以上のスペースで区切られた後には、そのオペコードに対するオペランドが続きます。二つ必要なときは“,”コンマで区切ります。このオペランドは、レジスタ名、16 進表現の定数、10 進表現の定数、フラグの状態(分岐の条件)名、を書きます。定数は、他の命令に付けたラベルを書いてもよく、ジャンプする場合などは特に絶対番地を計算しなくてもよいので便利です。オペランドのソースやディスティネーションに ( ) かっこを付けたときは、その内容を番地とするメモリの内容を意味します。

オペランドに書く定数は、10 進表現で書いてもよいし、16 進表現で書いてもよいのですが、16 進表現で書くときは、後に H (ヘキサデシマルの頭文字) を付けておきます。また、先頭がアルファベット A~F で始まるときはさらに前に 0 を付けてラベルでないことを特徴づける約束になっています。

コメントは行中のどこにでも“;”セミコロンがあれば、それ以後は自由にコメントエリアとして使うことができます。ただし 1 行は通常 80 文字位までで切られることがありますので、長くなる場合は数行に分けます。

Z 80 のアセンブリ言語は標準的な取り決めがありますが、アセンブラプログラムの種類(メーカや機種)により若干の違いがあります。アセンブラを使うときは、それぞれの説明書を一読してください。

Z80 MACRO ASSEMBLER V1.1 PAGE 1

```

1      ; IPL
2      ORG      0FF00H
3  FF00 3EED    LD      A,0EDH ;MODE
4  FF02 D3F3    OUT     (<0F3H>),A ;MDR0
5  FF04 3E37    LD      A,037H ;SLP
6  FF06 D3F2    OUT     (<0F2H>),A ;CDR
7  FF08 DBF6    LP0:   IN      A,<0F6H> ;ISR
8  FF0A CB7F    BIT     7,A ;CCE
9  FF0C 20FA    JR      NZ,LP0 ;CCE WAIT
10 FF0E 0610    LD      B,16 ;BLOCK COUNT
11 FF10 210000   LD      HL,0000H ;ADDRESS
12 FF13 3EFF    LP1:   LD      A,0FFH ;WDC SET
13 FF15 D3F1    OUT     (<0F1H>),A ;WDC
14 FF17 3E3B    LD      A,03BH ;RDL
15 FF19 D3F2    OUT     (<0F2H>),A ;CDR
16 FF1B DBF6    LP2:   IN      A,<0F6H> ;ISR
17 FF1D CB6F    BIT     5,A ;DA
18 FF1F 2007    JR      NZ,LP3 ;定数(番地)の代わりにラベル
19 FF21 DBF0    IN      A,<0F0H> ;DAR で指定
20 FF23 2F      CPL
21 FF24 77      LD      (HL),A ;DATA LOAD
22 FF25 23      INC     HL ;ADDRESS COUNT UP
23 FF26 10F3    JR      LP2
24 FF28 CB7F    LP3:   BIT     7,A ;CCE
25 FF2A 20EF    JR      NZ,LP2 ;CCE/DA WAIT
26 FF2C 10E5    DJNZ    LP1 ;END?
27 FF2E C300E0   JP      EXIT ;RETURN TO MONITOR
28      ; コメント行【見やすくするために一行あける】
29      EQU      0E000    EXIT: EQU 0E000H
30      END

```

ラベル オペコード

オペラント

マシンの語命令  
(オフセット)

ソースプログラム

タイプライタキーを打って入力する

アドレス

アセンブラが自動的に変換する

行番号

ORG: 以下のプログラムを配置するアドレスを指示する。

EQU: ラベル名の文字列をオペランドの数値とすることを指示する。

END: プログラムの終りを明示する。

## 命令の分類

**機能別**に命令を分類すると、次のようになります。

### 8ビットロード

1バイトの情報をレジスタ間、レジスタとメモリ間で転送する。1バイト定数をレジスタかメモリへ入れる。

### 16ビットロード

2バイトの情報をレジスタ間、レジスタとメモリ間で転送する。2バイト定数をレジスタへ入れる。

### レジスタ交換

レジスタの内容を入れ替える。

### メモリブロック転送

メモリ内の複数バイトのブロックを別のアドレスへ転送する。

### メモリブロックサーチ

メモリ内ブロックに指定の情報があるかどうか探す。

### 8ビット演算、論理演算

1バイト単位の加減算、論理和、論理積、排他的論理和、カウントアップ、カウントダウン、比較をする。1または2の補数をとる。

### 16ビット演算

2バイト単位の加減算、カウントアップ、カウントダウンをする。

### ローテートシフト

レジスタ、メモリのビットパターンを回転させ、または左行へずらす。

### 10進補正

2進法10進数として扱うデータの加減算後の補正をする。

### フラグ操作

キャリフラグを“1”にする、反転させる。

### CPU制

プログラムの実行と割り込みを制御する。

**ビット操作**

レジスタ、メモリの特定の1ビットを“0”か“1”にする、また、判定する。

**ジャンプ**

プログラムの実行を条件によりまたは無条件で、指定のアドレスへ移行させる。

**コール、リターン、リスタート**

プログラムの実行を条件によりまたは無条件で、指定のアドレスへ移行させる。ただしこのとき、リターン命令によりもとのアドレスへ戻れるような手順を含んでいる。

**入力出力**

指定のIOポートへレジスタとの間で1バイト情報を入・出力する。

**連続入出力**

メモリブロックを1バイト単位に連続して出力する。

メモリブロックへ1バイト単位に連続して入力する。

**マイナス数の表現（2の補数）**

2進数でマイナスを表現するには、多くの場合次の手法がとられます。

4ビットだけで考えれば、ゼロすなわち0000のひとつ前の-1は1111になります。なんとすればこれに1を加えると桁上りをして10000ですが、4ビットだけで考えているので桁上りを無視せざるを得ないからです。

		左端ビットが“1”のときを マイナス数と決める。	
0	0000	8	1000 → -8
1	0001	7	1001 → -7
2	0010	A(10)	1010 → -6
3	0011	B(11)	1011 → -5
4	0100	C(12)	1100 → -4
5	0101	D(13)	1101 → -3
6	0110	E(14)	1110 → -2
7	0111	F(15)	1111 → -1

上記のとおり、4ビットでは0～F(10)または-8～+7を表現することができます。正数を負数に変えるには、ビットの“1”、“0”を反転させて1を加えればよいのです。

## メモリ空間とIO空間

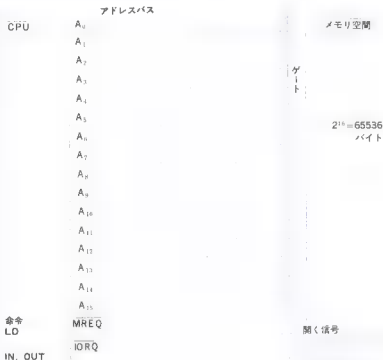
**Z** 80では接続できるメモリは最大64Kバイト(Kは1024)です。これはアドレスバスが16本あるので、 $2^{16}=64\text{K}$ となるからにほかなりません。メモリとは別にIOの領域として256のアドレスがあります。したがって、IOポートは256まで接続できることになります。これは、IOリード、IOライトサイクルではアドレスバスの下位8ビットのみ使用しているためで、 $2^8=256$ ということです。同じアドレスバスをメモリリクエスト信号とIOリクエスト信号により切り換えて、使用するためにメモリ空間とIO空間を別々に持ったことになります。

メモリ空間へのアクセスは、ロード[LD]命令を使い、IO空間へのアクセスは、イン[IN]、アウト[OUT]命令を使います。IO空間へのアクセスのほうが、タイミング的に長い時間がとれるようになっています。

アセンブラ記法でIO空間のアドレスを示す場合はOUT(OOH)、Aのようにかっこに入れる決まりになっています。

- バイト** ○ 8ビットを1バイトと呼ぶ。8ビットのCPUはバイトマシンと呼ばれ、8ビットを同時に扱うので、メモリは1アドレスのアクセスで8ビット(データバス8本)の読み書きが行える構成とする。
- したがって、1アドレス4ビットのメモリ(2114など)では2個、1アドレス1ビットのメモリ(4116など)では8個並列に並べる。





メモリリクエスト(MREQ)によりメモリがアドレスバスを受け入れるようにする。

IOリクエスト(IORQ)によりIOポートがアドレスバスの下位8ビットを受け入れるようにする。

◎実際にはアドレスバスはいつも受け入れておき、他の方法で回路なりLSIの応答を制御する場合が多い。

外部装置

## アドレスデコーダ

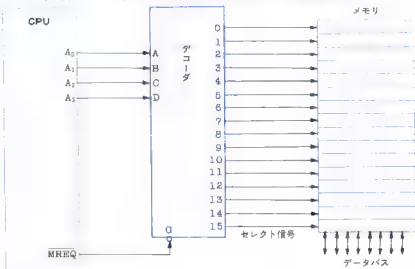
**ア**ドレスバスに出力されるビットパターンは、16本の端子から 65536 とおりもありますから、どのパターンのときはどのメモリを選ぶかを決めてやらなければなりません。

アドレスバスに出てくるアドレス信号のビットパターンから特定のメモリや IO ポートを選び出すセレクト信号（イネーブル信号）を作り出すわけです。メモリはたとえば、256 バイトのメモリであれば、LSI の中に 256 バイトすなわち、アドレスバス 8 本分のデコーダを持っていて、内部で特定のメモリセルを選択してくれます。ところが、このメモリを複数使用するときや、IO ポート用のペリフェラルを複数使用するときは、CPU とペリフェラルやメモリの間にアドレスデコーダが必要になります。デコーダは通常 74 HC シリーズなどの標準ロジック IC で構成します。

番地だけでなく、メモリ空間と IO 空間の切り換えも、ここで行なうのが普通です。メモリリクエスト ( $\overline{\text{MREQ}}$ ) がアクティブつまり、負論理出力ですから “L” になったときはメモリがイネーブルに、IO リクエスト ( $\overline{\text{IORQ}}$ ) がアクティブならば IO ポートがイネーブルになるようなロジックを組むのです。通常問題になることは少ないのですが、メモリや IO ポートへセレクト信号を出してから実際にイネーブルになり、データを受け入れまたは出力するまでの時間は、CPU の動作時間に適合しなければなりません。CPU より遅くて間に合わない場合は、ウェイト信号を CPU に与えて一時待たせるなどの方法をとります。メモリや IO ポートの応答時間と、アドレスデコーダなどの遅れ時間を含めて考える必要があります。しかし一般には、この遅れは無視できる場合のほうが多いようです。

アドレスデコーダとしてよく使われる標準ロジック IC は 74 HC 139 や 74 HC 155 などがあります。規格表から目的に合うものを探し出し、価格や入手状況を検討して決定してください。

## 16バイトメモリのアドレスデコーダ

16 バイト =  $2^4$  : 4本のアドレスバスで16のアドレスを表現できる

メモリの出力

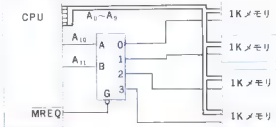
(Xは何でもよい)

G	D	C	B	A	出力
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	X	X	X	X	ナシ

- MREQ がノンアクティブ (負論理だから "1") のときはメモリセレクトはしない。
- MREQ がアクティブ ("0") のときだけ  $A_0$  から  $A_3$  のビットパターン16種により16本の出力のどれかへセレクト信号を出す。
- セレクトされたメモリはデータバスとデータの受け渡しを行う。

【参考】 1Kバイトのメモリ4個を使ったときのデコーダ

$$1K = 1024 = 2^{10}$$



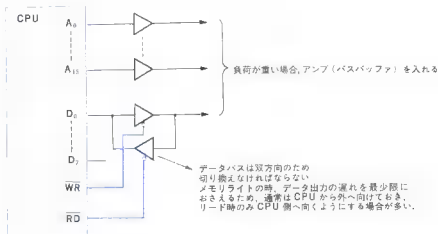
## バスバッファ

**T** TL のロジック IC には、ファンイン、ファンアウトという概念があります。マイコンの周辺の場合“H”と“L”のほかに、電氣的に切り離されたハイインピーダンスの3種類の状態をとるスリーステートの端子が多いため、単純には考えられなくなります。Z-80 ファミリの出力端子は、一つのノーマルタイプの TTL をドライブできる能力を持っています。LS タイプであれば四つまで可能です。これ以上の素子をドライブしなければならないときや、接続ケーブルを長く引き回すときは、バッファを入れなければなりません。特に長いケーブルを使用するときは波形の乱れやノイズ防止のため、バスドライバ、レシーバの能力に特別の考慮が望まれます。ときにはフォトカプラやシュミットトリガが使われることもあります。

メモリなどが何枚ものボードで構成される場合、ボードの出入口のところにバスバッファを設けるのが普通です。単にバッファだけでなく、ゲートの働きもして、そのボード内のアドレスがアクセスされたときだけゲートが開くようにしておけば、CPU に対して無駄な負荷をかけることもなく、デバッグもやりやすくなります。

データベースに使用するバッファは双方向性です。リード、ライトの信号によって方向を決定します。マルチ CPU システムとしたときや、DMA を使用したときはきわめて複雑になりますので、システムのブロック化をするときには十分考慮する必要があります。

割り込みを使用するシステムでは、ペリフェラルは、CPU がメモリからフェッチしてくる命令を横からデータベースを見ていて、割り込みからの復帰を知るようになっています。したがって、データベースのバッファはフェッチサイクルではメモリから CPU へ向くと同時に、メモリからペリフェラルへも送り込まなければなりません。



- ゲートはユニットの機能により、開け閉めする場合もあるが、常時開けておき、他のゲートを制御するための情報を受けとる場合もある。

## システムクロック

**Z**-80 ファミリは、システムのすべての動作はシステムクロックに同期して進められますので、CPU やペリフェラルのクロック ( $\phi$ ) 端子へ共通のクロックパルスを与えなければなりません。当然周波数が高いほど処理速度は速いわけですが LSI の性能で上限は規定されます。また周辺に接続されるメモリや入出力の端末の応答速度が追従しなければ、なんの意味もなく、単に CPU を待たせる回路が増えるだけになります。

システムクロック ( $\phi$ ) は単相の方形波ですが、“H”と“L”の幅がそれぞれ規定されています。したがって、周波数が規定以下であっても上限に近いときはデューティ比が 50% に近くないと具合がわるい場合があります。そのときは目的のクロック周波数の 2 倍の周波数で発振させ、1/2 分周すれば正確にデューティ比 50% のパルスが得られます。

クロックパルス 1 周期を**クロックサイクル**と呼びます。マシンサイクルは、クロックサイクル数個によって一つずつ進められます。1 クロックサイクルは、クロック周波数の逆数 [秒] ですから。たとえばクロック周波数が 4 MHz なら  $1/(4 \times 10^6) = 0.25 \mu\text{s}$ 、8 MHz なら  $1/(8 \times 10^6) = 0.125 \mu\text{s}$  となります。



## リセット

**C**PUのリセット(**RESET**)端子へのリセット信号は、パワーオン時には与えなければなりません。CPU以外にも同じ信号を与えます。PIOとDMAはパワーオンリセットの機能を内部に持っていますので必要ありませんが、与えてもよいのです。リセットの時間は、電源電圧が安定し、クロックが安定に与えられてから最低3クロックサイクル分(2.5MHzクロックであれば、 $1/(2.5 \times 10^6) = 0.4 \mu\text{s}$ 。したがって、 $0.4 \times 3 = 1.2 \mu\text{s}$ )以上です。実用上問題なければ、長いほど安全といえます。

CPUは、このリセット信号により

プログラムカウンタ(PC)

Iレジスタ

Rレジスタ

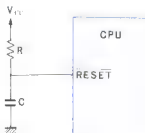
をゼロにして、割り込みモードを0として、割り込みを受け付けない(インタラプトディスエーブル)状態にします。これ以外のレジスタ(もちろん、CPU外のRAMも)は変化しません。

リセット信号がノンアクティブ(負論理ですから“H”)になると、まず、ゼロ番地の命令をフェッチすることから動作を始めます。もしIレジスタを使用するなら、ゼロ番地からのプログラムでIレジスタへ必要な数値を書き、割り込みモードを希望のモードに設定し、その他必要な初期化を行ってから、割り込みを受け付け可の状態(インタラプトイネーブル)にする(EI)命令を実行させます。またスタックポインタ(SP)の設定もしなければなりません。

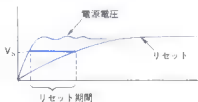
リセット信号は、パワーオン時に与えることはもちろんですが、プログラムが意図しない無限ループに入ってしまうこともあり、特にデバグ中には、手動で電源をオンオフせずにリセットできるようにしておく大変便利です。ただし、なんらかの動作中に不用意にリセットすると困ることもありますので注意が必要です。リセット期間中はメモリリフレッシュ信号は出ません。



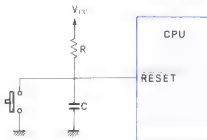
## パワーオンリセット



RとCにより数msの待込数を持たせる



## マニュアルリセット



パワーオンリセットも兼ねる。スイッチのチャタリングはCにより吸収される。

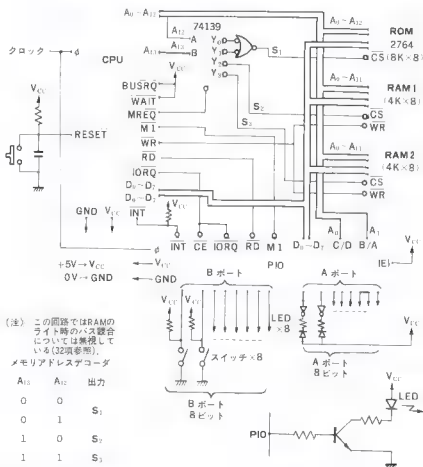
- \* リセット信号にノイズがのることを想定すると、シュミットトリガ回路を通した方が確実といえる。

## システム構成

マイコンシステムの構成は、目的によって大きく異なります。普通は CPU 1 個に対して、入出力のポート数あるいはビット数を満たすだけの IO ベリフェラル、そしてプログラムを取容しきれるだけの ROM と、データエリア（ワーキングエリア）としての RAM、IO とメモリのアドレスデコード、クロック、リセットで構成されます。さらに大きなシステムでは DMA を採用したり CTC を追加したり、場合によっては CPU を複数にしたマルチ CPU システムとすることもあります。マルチ CPU システムは、目的の仕事がはっきり区分できるときはプログラムが楽になり、効率がよくなる点でメリットがあります。

図は入力にスイッチ 8 個、出力に LED 8 個、プログラムは 8K バイト以下、データエリアも 8K バイト以下、といった簡易なコントローラの例を示したものです。メモリは合計 16K バイトですので、アドレスバスは下位 14 ビットだけを使い、上位 2 ビットはあそばせてあります。したがって、14 ビットだけが有効で上位桁は何であってもよいわけです。何であってもよいということは、何であってもいずれかのメモリがセレクトされてしまうので、これ以上増設することはできません。IO ポートには PIO を 1 個使っておりますので、アドレスは四つだけ必要（詳細は PIO の項参照）になり、下位 2 ビットだけ有効です。

このシステムでもスイッチをマイクロスイッチや光センサに、LED をリレーやソレノイドに変えれば、メカニズムのコントロールロジックを置き換えたり、簡単なシーケンスコントローラとして実用化することができます。



各メモリ LSI の中でデコードしてくれ目ので  
最小値と最大値のみを記入した

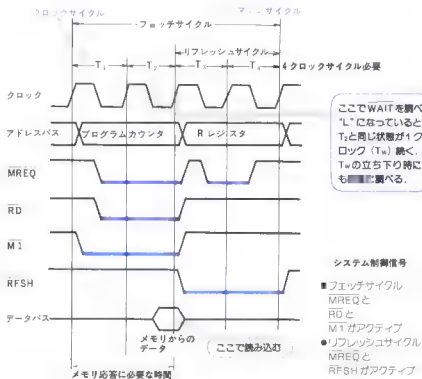
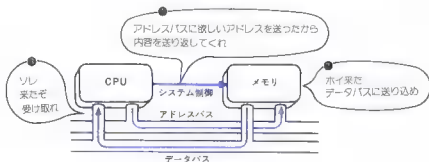
## フェッチサイクルの動作

— つの命令を実行するとき、最初に CPU はフェッチサイクルに入ります。プログラムカウンタの値に示されるアドレスのメモリからオペコードを読み込み、解析します。そしてプログラムカウンタに1を加え、次の読み込みに備えます。

フェッチサイクルが始まるとアドレスバスにプログラムカウンタの値が乗せられ、同時にエムワン ( $\overline{\text{M1}}$ ) 信号が“L”すなわちアクティブになります。アドレスバス上のデータの安定するのを待ってメモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号が、続いてリード ( $\overline{\text{RD}}$ ) 信号がアクティブになってメモリをアクセスします。3 番目のクロック  $T_3$  の立ち上がりで、CPU はデータバスの内容を読み込み (サンプリング) します。 $T_2$  が立ち下がる時にウエイト ( $\overline{\text{WAIT}}$ ) 信号がアクティブになっていると、そのままの状態でもウエイトサイクル  $T_w$  が  $T_2$  と  $T_3$  の間に入ります。 $T_w$  の立ち下がりでもウエイト ( $\overline{\text{WAIT}}$ ) 信号を調べますので、次へ進めたいときはウエイト ( $\overline{\text{WAIT}}$ ) を、ノンアクティブ “H” に戻さなければなりません。応答の遅いメモリを使うときに、これで時間待ちさせることができます。 $T_3$  では DRAM をリフレッシュするためのリフレッシュアドレスがアドレスバスの下位 7 ビットに出され、リフレッシュ ( $\overline{\text{RFSH}}$ ) 信号も同時にアクティブになります。メモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号はアドレスバスが安定しているであろう期間にアクティブになります。

リフレッシュ中は、CPU の内部では命令の解析が行われ、次のサイクルに何をすべきか判断していきます。8 ビットの汎用レジスタ間の転送命令などでは 4 クロックのフェッチサイクルだけで実行を完了してしまいます。

フェッチサイクルのメモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号は、データバスをサンプリングする約 1.5 クロック前に出されますので、この時間内にメモリが応答しなければなりません。しかし問題になることは少なく、万一のときもウエイト ( $\overline{\text{WAIT}}$ ) 信号で解決できます。



## メモリリードサイクル

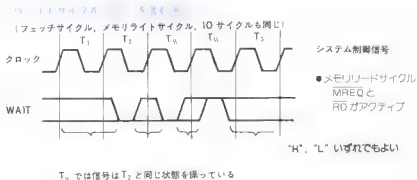
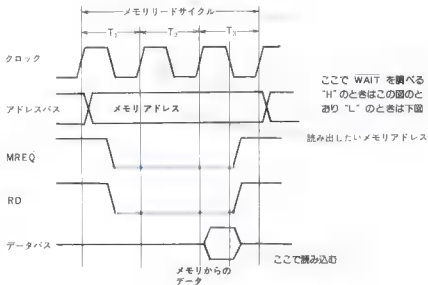
メモリからデータを読み出して、レジスタや他のメモリへ転送する命令の1サイクルとして、メモリリードサイクルへ入る場合と、命令語のオペランドを読み出すためのメモリリードサイクルがあります。動作はどちらも同じで、通常<sup>※1</sup>3クロックサイクルの間に実行されます。

アドレスバスに読み込むべきアドレスが送出され、これが安定するのを待ってメモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号とリード ( $\text{RD}$ ) 信号が同時に出力されます。2番目のクロックは、メモリが応答し、データバスへ情報を乗せてくるのを待つ時間です。もしメモリの応答が間に合わない場合には、立ち下がり前後にウェイト信号を入れることができます。3番目のクロックの立ち下がり時点に、CPUはデータバスのゲートを開き、内部へデータを読み込みます。

リード ( $\text{RD}$ ) 信号とメモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号が出てからCPUが読み込むまでの時間は、フェッチサイクルでは1.5クロックでしたが、メモリリードサイクルでは2クロックあります。クロックを2.5MHzとすると、1クロック当りの時間は400ns<sup>※2</sup>ですから、フェッチサイクルのときは600ns、メモリリードサイクルのときは800nsとなります。したがって、アクセスタイムが450ns以下のメモリであれば問題なく使用できます。

■ 1 命令により例外はある。

※ 2 ns：ナノセックといい、 $10^{-9}$ 秒。



## メモリライトサイクル

**C**PUからメモリに対する書き出しのマシンサイクルです。レジスタからメモリ、メモリからメモリへの転送命令などで実行されますが、サブルーチンコールのコール命令で、戻り番地をスタックへ格納するときや、割り込みがなかったときも実行されます。いずれも同じタイミングで通常\*3クロックです。前項のメモリリードサイクルとはほとんど同じで、異なるのはリード (RD) 信号が出ずに、ライト (WR) 信号が出る点と、CPUからデータバスへデータが乗せられる点です。

ライト (WR) 信号は、メモリリクエスト (MREQ) 信号より1クロック遅れて出されます。これは、データバスが安定するのを待つためで、ライト (WR) 信号でメモリの読み込みのゲートをコントロールすることができるようになっています。

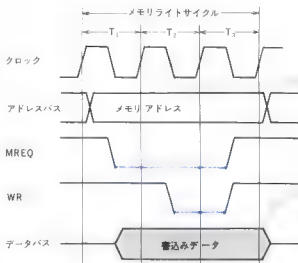
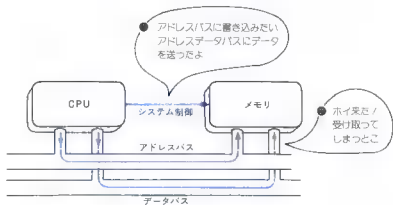
### メーカーの立場とユーザの立場

マイクロプロセッサは、一つの電子部品との見方からすると、その使い方は、ユーザの責任において研究されなければなりません。汎用のICやトランジスタでも、メーカーの発表するのは一使用例にすぎず、回路構成や応用製品の信頼性については、すべてユーザの研究や雑誌などのレポートを参考にして使用されてきました。

コンピュータと呼ばれることから、大型コンピュータ並の指導やサービスをメーカーに期待すると、あてがはずれます。LSIの単価には、これらの費用は含まれてはいないからです。マニュアルも有償である場合がほとんどです。マニュアルは、指図書ではありませんので、わかりにくいものです。基本的に「何をどうするとどうなります」というLSIの機能を述べ連らねてあるにすぎません。書物や雑誌に目を向け、いろいろな事例に接することが大切です。

■ 命令により例外はある。





ここで WAIT を開ける

WAIT については  
メモリリードサイ  
クルと同じ

CPU からのデータはこの  
データバスに出されている  
WR 信号によってメモリへ入る  
ようにすればよい

システム 信号

メモリライトサイクル  
MREQ と  
WR がアクティブ

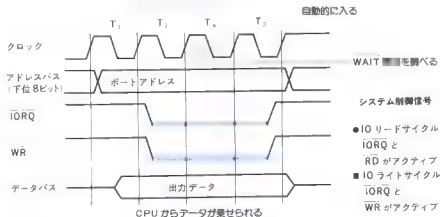
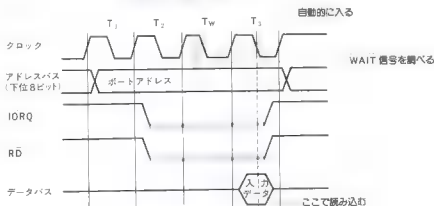
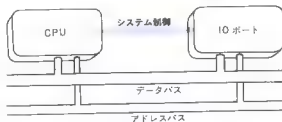
## IO リードサイクル, IO ライトサイクル

I Oポートに対するリード、ライトは、アドレスバスの下位8ビットが有効なアドレスとして使われます。上位8ビットへはなんらかの状態が出力されますので、無視しなければなりません。

イン〔IN〕命令、アウト〔OUT〕命令と、これらのくり返しの命令を実行したときに、このサイクルへ入ります。

IO サイクルでは、メモリの読み、書きより長く時間を必要とすることが想定されますので、ウェイトサイクル  $T_w$  が自動的に入ります。もっと長く時間待ちさせたいときは、この  $T_w$  の立ち下がりの前後にウェイト (WAIT) 信号を入れることになります。またリード (RD)、ライト (WR) および IO リクエスト (IORQ) の各信号は、立ち下がり、立ち上がり共に同じタイミングです。

イン〔IN〕命令、アウト〔OUT〕命令は、入出力するポートアドレスを指定しなければなりません。直接指定の場合は A レジスタとの間でのやりとりになり、ポートアドレスを  $n$  とすると〔IN A, (n)〕,〔OUT (n), A〕と書きます。間接指定の場合は、ポートアドレスを C レジスタに入れておき、〔IN r, (C)〕,〔OUT (C), r〕(ここで  $r$  は A, B, C, D, E, H, L のいずれかのレジスタ)と書きます。このほかにメモリの一連の複数バイトのブロックを入出力するブロック入出力命令があります。

この間のやりとり  
を行う

## リフレッシュサイクル

リフレッシュサイクルは、各フェッチサイクルの後半に行われます。アドレスバスの下位7ビットには、Rレジスタの内容が出力されます。Rレジスタの内容は1回ごとに1ずつ増やされ、DRAMの1列ごとに再書き込みを行います。

このときはリード( $\overline{RD}$ )、ライト( $\overline{WR}$ )の信号の代わりにリフレッシュ( $\overline{RFSH}$ )信号が出て、リフレッシュサイクルであることを知らせます。

リフレッシュは、長くても2ms以内に1回以上行わなければならないので、Z-80を最高速で働かせたときでも、16KビットのDRAMが限界になります。16KビットのDRAMの代表的なものに、4116と呼ばれるものがありますが、 $16K=2^{14}$ で、アドレスバスは14ビット必要です。ところが上位、下位7ビットずつを時分割して与えますので、アドレス入力は7端子しかありません。リフレッシュの場合は、7ビットのアドレスだけを順次与えればよいのです。したがって、Rレジスタは7ビットとなっています。

この機能はZ-80が開発されたころ、主流であった16KビットDRAMを対象に考えられたため、現在使われるメガビットクラスのDRAMでは、Z-80の機能とは別にリフレッシュ回路を作らなければなりません。

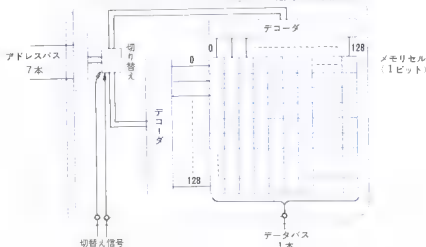
### MOS LSI の取り扱い

Z-80ファミリに限らず、MOSプロセスのLSIは、端子のインピーダンスが高いため帯電しやすく、チップの絶縁が破壊されることがあります。保存するときはモスパッドと呼ばれる導電性のスポンジにさしてピン間をショートしておけばまちがいがありません。また、ピンに直接手で触れたり、帯電していそうな所に接触しないように注意しなければなりません。はんだごても絶縁のよいものか。アースのできるものを使用します。実際には、そう簡単にこわれてしまうわけではありませんが、念のために心がけたほうがよいでしょう。

DRAMの模式図・4116



リフレッシュ  
ゼロになってしまう前に（1であることが判別できる  
うちに）1であれば再び1を書き込む



- アドレスバス 7本へ 上位 7 ビット、下位 7 ビットを時分割で与える。
- いま、上位が与えられているか、下位が与えられているかを切替え信号で知らせる。
- リフレッシュの場合は、一方だけを与え、他方は全メモリセルをセレクトする。→ データ出力は無意味（デタラメ）→ データ出力なし。

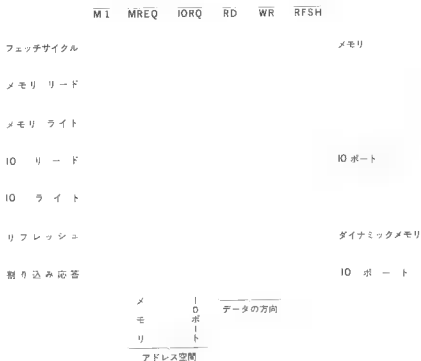
## CPU と周辺の接続（インターフェース）

**各** マシンサイクルごとにアドレスバスとデータバスの意味が変わりますので、どの動作をしているかによって、メモリ空間と IO 空間のどれを選ぶか、また、書き込みか読み出しかの切り換えをしなくてはなりません。

まず、アドレスバスをデコードして、セレクト信号を作ります。次にメモリリクエスト（ $\overline{\text{MREQ}}$ ）と IO リクエスト（ $\overline{\text{IORQ}}$ ）信号によりメモリ空間と IO 空間を切り替えます。ここで IO リクエスト（ $\overline{\text{IORQ}}$ ）信号はエムワン（ $\overline{\text{M1}}$ ）信号と同時にアクティブになったときは別の意味（割り込み応答）になりますので、このときは IO 空間をセレクトしてはいけません。Z-80 ファミリではセレクトしても問題ありません。

IO ポートに Z-80 ファミリの LSI を使うときは、同じ名称のピンどうしを接続すれば、システム制御信号に関してはなんら心配はいりません。2 個以上の LSI を接続するならば、アドレスデコードしたチップセレクト信号をそれぞれの LSI に与えなければなりません。

メモリは種類によって異なります。ROM の場合は、チップセレクト信号は、リード（ $\overline{\text{RD}}$ ）信号が出たときだけ与えるようにしなければなりません。RAM の中にはアウトプットドライブ（ $\overline{\text{OD}}$ ）端子があるものがありますが、この端子へはリード信号を直接与えます。ライトイネーブル（ $\overline{\text{WE}}$ ）端子へはライト（ $\overline{\text{WR}}$ ）信号を与えればよいのです。アウトプットドライブ（ $\overline{\text{OD}}$ ）端子がない RAM の場合は、ライトイネーブル（ $\overline{\text{WE}}$ ）端子へ与える信号には工夫が必要です。メモリライトサイクルでは、メモリリクエスト信号が出て少したってからライト信号が出ますが、このライト（ $\overline{\text{WR}}$ ）信号が出るまでの時間、メモリは読み込み動作をしてしまい、CPU からのデータとメモリからのデータがデータバス上でぶつかってしまいます。実際問題としては LSI をこわしてしまうほどのことはないようですが、さけるべきです。リード（ $\overline{\text{RD}}$ ）信号とリフレッシュ（ $\overline{\text{RFSH}}$ ）信号がなく、かつ、メモリリクエスト（ $\overline{\text{MREQ}}$ ）信号が出たときだけライトイネーブル（ $\overline{\text{WE}}$ ）端子へ与える信号を作れば解決できます。



- この信号の組合せでメモリやIOポートをイネーブル（活かす）したり、デイスエーブル（殺す）したりする（ように接続する）。アドレスデコーダをコントロールして、セレクト信号が出るようにしたり、止めたりしてもよい。
- Z-80 ファミリで構成するときは、同じ信号線どうしを接続すればよい。

## ウェイト信号とホルト信号

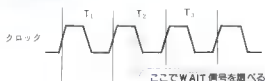
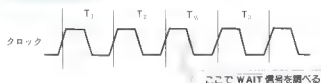
**ウェイト (WAIT) 信号**は、メモリや IO の応答速度が CPU のタイミングに間に合わないときに、タミーのクロックサイクルを入れて、CPU を待たせるときに与えます。CPU は各マシンサイクルの 2 番目のクロックの立ち下がりの時点で、ウェイト信号端子の状態を検知して、このとき“L”になっていれば、次のクロックをウェイトサイクルとして何もしないで待ちます。このウェイトサイクルの立ち下がりのときも同じく検知していますので、“H”になるまで空転していることになります。ウェイト中はリフレッシュは行われません。

IO ライトサイクルと IO リードサイクルでは一つ、割り込み応答のフェッチサイクルでは二つのウェイトサイクルが自動的に挿入されます。

簡単なシステムではウェイトの端子を使用しないことのほうが多いと考えられますが、そのときは  $V_{CC}$  へ続いて“H”レベルにしておかないと、CPU は動作したり止まったり、不安定な状態になってしまいます。

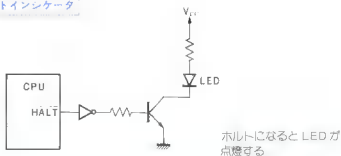
**ホルト (HALT) 信号**は、CPU がホルト (HALT) 命令を実行したときに出力される信号です。[HALT] 命令を実行すると、プログラムカウンタの進行を止めてしまいますので、CPU が停止したのと同じです。普通のプログラムでは、このようなことは起こり得ませんので、異常があったことを外部へ知らせる信号として使えます。ホルト状態からの解除は、リセット信号を与えるか、割り込み受け付け可になっているときは、割り込みをかければよいのです。ホルト状態では、ノンオペレーション (NOP) 命令といって、フェッチサイクルだけで何もしない命令をくり返し実行しています。このようにしてある理由は、ホルト中もメモリリフレッシュを絶やさないためです。



T<sub>1</sub>がない場合T<sub>1</sub>がある場合

- $T_2$  の立ち下り、または  $T_W$  の立ち下りで WAIT を調べ、  
“H” ならそのまま、“L” なら  $T_W$  を挿入する。
- 挿入された  $T_W$  の立ち下りでも WAIT を調べ、“H” なら次は  $T_3$ 、“L” ならもうひとつ  $T_W$  を挿入する。
- WAIT が “H” になるまで何回でも  $T_W$  が入る。

## ホルトインシケータ



## 割り込みの概念

CPU は、リセット (RESET) 信号が入ると、ゼロ番地から順次命令を実行します。命令の中に、「ある番地へジャンプせよ」という命令、ジャンプ (JP) 命令やコール (CALL) 命令があると、その番地の命令を実行します。つまり、通常はプログラムカウンタを一つずつくりあげていき、特別の命令があると、プログラムカウンタの内容をその命令に従って変更します。

いま、このようにして処理を進めているときに、急に他の処理をしなければならなくなったことを考えてみます。停電になってとりあえずバッテリーに切り換えるとか、端末装置から故障を知らせる信号が入ったときに、通常のプログラムの実行を止めて、対策処理をしなければならないことがあります。このようなときには、インタラプト (INT) 信号かノンマスカブルインタラプト (NMI) 信号を入れることによって、プログラム上の命令とは関係なく、特定の番地へジャンプさせることができます。これを**割り込み**と呼びます。

割り込み機能を積極的に使うことにより、処理効率を高めたり、プログラムを簡単にしたりすることができます。たとえば、CPU からプリンタにデータを打ち出すよう命令を与えてからプリンタが動作を終えるまでの時間は、CPU の処理速度に対して非常に長いのですが、この間ただ待つのではなく、CPU は別の仕事をしています。プリンタからの動作終了割り込みで次のデータを打ち出すことができます。一度に二つの仕事をしているように見えるのでマルチタスクとかマルチジョブと呼んで、入出力の多い仕事では CPU の処理時間は無視され、端末装置の動作時間だけで処理速度がほぼ決まってきます。

割り込みは CPU の動作に同期せずに、外部からのハード的な要求で呼び出されるサブルーチンコールと考えてよいでしょう。



## ノンマスカブルインタラプト (NMI)

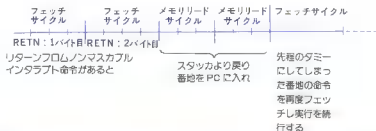
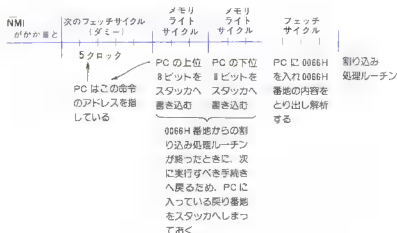
**Z**-80には、ノンマスカブルインタラプト (NMI) と、ただのインタラプト (INT) の2系統の割り込みがあります。ノンマスカブルインタラプトは、いかなるときでも受け付ける最優先の割り込みです (ただし、バスアクリッジ中は受け付けません)。したがって、停電などシステムやオペレータにダメージを与えるような非常時の対策に使うことが多いようです。

ノンマスカブルインタラプトは名前のとおり、プログラムによってマスク、すなわち無効にすることができない割り込みです。CPUはノンマスカブルインタラプト (NMI) 信号が立ち下がったとき、実行中の命令が終わり次第、割り込み処理に入ります。割り込みの処理プログラムが終わったとき、また元の番地へ戻らなければならないために、現在のプログラムカウンタ (PC) の内容をスタックへ格納します。次に、プログラムカウンタへ 0066H を書き込み、新しいフェッチサイクルに入ります。したがって、0066H 番地をコールすることになります。割り込みがかかった時点で実行中の命令は終わりまで実行し、次の命令のフェッチサイクルもあたかも平常のとおりに実行されますが、データバスを無視して (CALL 0066H) をフェッチしたかのように動作を続けていると考えられます (ただし、タイミングは異なります)。0066H 番地には割り込み処理プログラムを書いておかなければなりません。そして処理の最後にリターンfromノンマスカブルインタラプト (RETN) 命令があれば、スタックへ格納しておいた戻り番地をとり出して、割り込みがかかったときに実行していた次の命令へ復帰します。ノンマスカブルインタラプト (NMI) が入ってからリターンfromノンマスカブルインタラプト (RETN) が実行されるまではインタラプト (INT) は無視され、待たされることになります。



これより前に NMI の立ち下りがあると

次のフェッチサイクルは読み込んだ命令を無視してプログラムカウンタの内容をスタックへ入れ 0066H 番地へジャンプする



## インタラプト (INT)

リセット信号が入ってCPUが初期状態から実行し始めると、インタラプト信号に対するマスクは、ディスエーブルすなわち割り込みを無視するようになっています。また割り込みモードは0になっています。もし割り込みモード1か2で使用するのなら、IM1かIM2の命令を実行させておく必要があります。割り込みモード2の場合は、Iレジスタとベリフェラル内のレジスタに対し数値を設定しなければなりません。これらの準備が終わったところで、割り込みを許可する命令、イネーブルインタラプト(EI)命令を実行させますと、これ以降割り込みがかかるようになります。割り込みを禁止するときは、ディスエーブルインタラプト(DI)命令です。

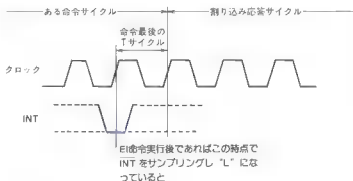
割り込みがかかると、次の割り込みがかからないようにディスエーブルインタラプト(DI)命令を自動的に実行し、おのおのの番地へジャンプします。割り込み処理プログラムの終わりには、リターン(RET)命令が書かれていれば、CPUは元の処理を続行するべく戻り番地へジャンプします。ただしこのとき、リターン(RET)命令の直前にイネーブルインタラプト(EI)命令を置かなければ、割り込みは禁止されたままになってしまいます。イネーブルインタラプト(EI)はただちに有効になるのではなく、次の1命令を実行し終わったときから有効になりますので、リターン(RET)命令が終了した後に割り込みを受け付けるようになります。

なお、ノンマスクابلインタラプトは信号の立ち下がりエッジでかかりますが、インタラプトのほうは各命令の最後のクロックサイクルの立ち上がり時点で信号の状態を検査しますので、このとき“L”になっていなければなりません。

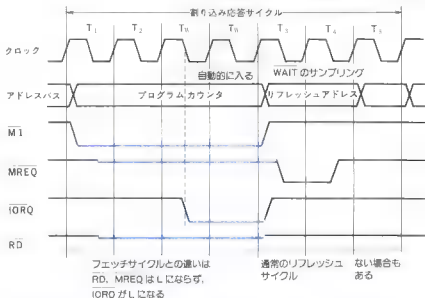
Z-80ファミリのベリフェラルは、優先順位を決定する機能を個々に持っており、外部に回路を必要としません。この機能を使うときは、割り込み処理が終わったことを通常のサブルーチンからのリターンとは区別してベリフェラルに与えなければならないため、リターン(RET)命令ではなくリターンフロムインタラプト(RETI)命令を使用します。

IM 0	割り込みモードを設定する命令。いずれかのモードを選びプログラムの最初の方
IM 1	で1回実行させる。
IM 2	
EI	割り込み信号を許可する命令
DI	CPU 側で割り込み信号を受け付けなくする命令

EIを実行後 CPU は割り込みを受け付け、DIを実行すると受け付けなくなる。



次のフェッチサイクルは割り込み応答サイクルになる

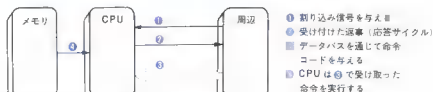


## モード0のインタラプト

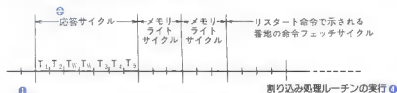
**割**り込みモード0は、命令の最後のクロックサイクルの立ち上がりのとき、インタラプト ( $\overline{\text{INT}}$ ) 信号が“L”になっていると、次のフェッチサイクルは割り込み応答サイクルとなります。フェッチサイクルとの違いは、メモリリクエスト ( $\overline{\text{MREQ}}$ ) 信号が出ずにIOリクエスト ( $\overline{\text{IORQ}}$ ) 信号が出力されることと、3, 4番目のクロックサイクルにウエイトサイクルが2個、自動的に挿入されていることです。したがって、ペリフェラル側では、エムワン ( $\overline{\text{MI}}$ ) 信号とIOリクエスト ( $\overline{\text{IORQ}}$ ) 信号が共にアクティブ、すなわち“L”になったことで、割り込みがかかったことを知ります。割り込み応答サイクルでは、アドレスバスには次の命令のアドレス、すなわちプログラムカウンタの値が出ますが、外部からはこれとは関係なく、なんらかの命令をデータバスに乗せてやります。この命令は主にリスタート [ $\text{RST}$ ] 命令か、コール [ $\text{CALL}$ ] 命令を使います。リスタート ( $\text{RST}$ ) 命令は1バイト構成ですからこれでよいのですがコール [ $\text{CALL}$ ] 命令のときは3バイト構成です。そのため、あと2回メモリリードサイクルに合わせて、データバスへ割り込み原因に応じたプログラムルーチンを実行させるための命令の続き(コール命令のジャンプ先番地を意味するオペランド)を与えなければなりません。

CPUの読み込みタイミングは、フェッチサイクルと同じく  $T_0$  の立ち上がりです。またリスタート ( $\text{RST}$ ) 命令を与えたときは、スタックポインタの変更をするためのクロックサイクル  $T_0$  が付きますが、コール [ $\text{CALL}$ ] 命令ではこれがなく、マシンサイクルの3番目 ( $\text{M3}$  サイクル) か、メモリリードサイクルの3クロックの後に  $T_0$  として続き、4クロックサイクルとなります。これは通常の場合にフェッチサイクルでコール [ $\text{CALL}$ ] 命令やリスタート [ $\text{RST}$ ] 命令をフェッチしたのと同じです。すなわち応答サイクルで、アドレスバスを無視して強制的にデータバスへ命令を与えれば、それ以後は通常その命令を実行するのと同じ動作をしています。





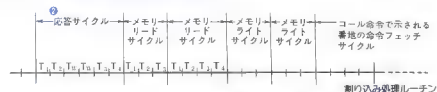
## リスタート RST 命令を与えた場合



PCの上位を PCの低位を  
スタックへ スタックへ

この時点でCPUはアータバス上の  
命令を読み込む (RST 命令)

## コール CALL 命令を与えた場合



PCの上位を PCの低位を  
スタックへ スタックへ

コール命令の3バイト目  
(ジャンプ先番地の上位)  
を読み込む

コール命令の2バイト目  
(ジャンプ先番地の下位)  
を読み込む

この時点でCPUはアータバス上の  
命令を読み込む (CALL 命令)

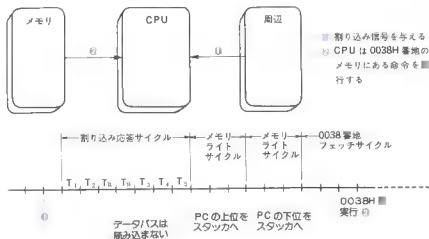
このタイミングに合わせて  
周辺からデータバスに乗せ  
る

## モード 1 のインタラプト

**割**り込みモード 1 は最も簡単な割り込みで、ほとんどノンマスカブルインタラプトと同じ動作をします。

インタラプト (INT) 信号の検知と、応答サイクルは割り込みモード 0 と同じです。応答サイクルでは、外部からは何も与えず、CPU も何も読み込まないので、デミーサイクルとなります。次に現在のプログラムカウンタの値、すなわち戻り番地をスタックへ格納するためにメモリライトサイクルが 2 回続き、そのあとプログラムカウンタを 0038H にします。次はこの番地の命令の実行サイクルへ入ります。見かけ上は、(CALL 0038H) を実行したことになります (ただし、タイミングは異なります)。

このモードでは、割り込み要因によって、ジャンプ先の処理ルーチンを分けることはできません。もしこれが必要なときは、0038H 番地からのルーチンの中で、外部の状況を読み込み、解析しなければなりません。



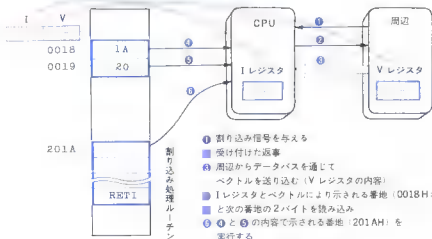
## モード2のインタラプト

**割**り込みがかけられた次のフェッチサイクルは、割り込み応答サイクルとなります。他のモードと違うのは、エムワン ( $\overline{M1}$ ) 信号と IO リクエスト ( $\overline{IORQ}$ ) 信号が共に “L” になったら、ペリフェラルからはベクトルと呼ばれる 1 バイトの値をデータバスに乗せ、CPU がこれを受け取ることです。次に戻り番地として現在のプログラムカウンタの値をスタックへ格納するためのメモライツサイクルが 2 回続きます。その次のサイクルでは、CPU は 2 バイトの情報をメモリから読み込むためのメモリリードサイクルが 2 回続きます。そしていま読み込んだ値をプログラムカウンタへ入れて、そのアドレスからの実行を開始します。

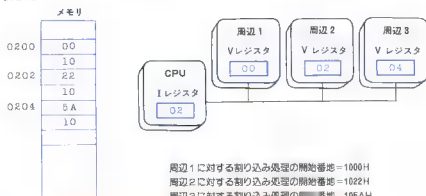
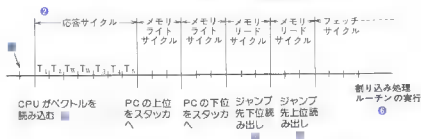
**ベクトル**とは、各ペリフェラルにあらかじめ書き込んである 1 バイトのデータで、これがアドレスの下位 8 ビットとなります。上位 8 ビットは、CPU 内の I レジスタへ、これもあらかじめ書き込んでおきます。割り込みがかかると、I レジスタの内容とベクトルで構成される番地と、次の番地のメモリの内容を 2 バイト読み込んで、この値の番地へジャンプすることになります。

ベクトルの値は、各ペリフェラル個々に異なった値を書いておけば、割り込みをかけたペリフェラルによって、別々の処理ルーチンへジャンプしますし、DMA や SIO では、割り込み発生の原因によってベクトルを変化させて送り出しますので、処理ルーチンの組み立てが簡単にできます。

このモード2の割り込み機能は Z-80 の大きな特徴です。ファミリと合わせてシステムを構成すれば、外部回路なしで優先順位の決定を含めた、割り込みシステムが完成してしまいます。



(上記の番地の数値は一例でありこの関係を保てば全メモリのどの番地でもよい)



周辺1 に対する割り込み処理の開始番地 = 1000H  
 周辺2 に対する割り込み処理の開始番地 = 1022H  
 周辺3 に対する割り込み処理の開始番地 = 105AH

このような表を割り込み処理ルーチンエントリ・リードアドレステーブルと呼ぶ

## デージーチェーン

**割**り込みの優先順位の決定にはデージーチェーン（ひな菊の輪）という手法がとられます。CPU に対して割り込み信号を出すペリフェラルを直列につないでおき、割り込みをかけて処理ルーチン実行中のペリフェラルが、自分より下位のペリフェラルに対して割り込み禁止を順次伝達していきます。自分より上位の割り込みがかかると、自分の処理が保留されますので、これも、デージーチェーンを通じて知ることができます。処理が終わってリターンフロムインタラプト (RETI) 命令を CPU がフェッチすると、データバスを横から監視している処理中のペリフェラルは自分の処理が終わったことを知り、下位のペリフェラルに対して割り込みの禁止を解除します。

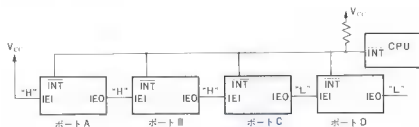
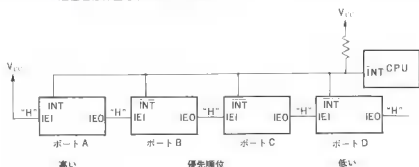
各ペリフェラルはインタラプトイネーブルイン (IEI) とインタラプトイネーブルアウト (IEO) 信号の端子を持っており、IEO を自分より下位の IEI につなぎます。最も優先順位の高いペリフェラルの IEI は  $V_{CC}$  (+5V) へつなぎ、最も低いペリフェラルの IEO はオープンにしておきます。

多数のペリフェラルをつなぐときは、伝達に時間がかかりますので、工夫があります。そのままでつなげるのは4個までです。

この方式は、割り込みだけでなく、マルチ CPU としたときや、DMA を複数使用するときのバスリクエストでも使われます。

## 入 力 IEI 出 力 IEO の 操 作

- ① 通常 IEI が "H" になっていれば IEO も "H" が出ている。
- ② 割り込みを発生し ( $\overline{\text{INT}}$ ) 応答サイクルに入ると、割り込みを発生したペリフェラルは IEO を "L" にする。
- ③ IEI に "L" が入ると IEO も "L" にする。
- ④ 割り込みは、IEI が "H" になっているときだけ発生し、"L" のときは発生しない、またすでに発生した後に IEI が "L" になったときは保留されたものとする。

割り込み発生  
応答受動

このときも割り込み要因があれば CPU に対して割り込みを発生できる。  
このときは優先順位の低いポートは割り込み中であっても保留させられ、高いほうを先に処理する

割り込み処理終了の命令 [RETI] を CPU がフェッチしたことをデータバスを見ていて知ると、IEO を "H" に戻す

これより優先順位の高いポートが割り込み中なので終わるまで待たされる

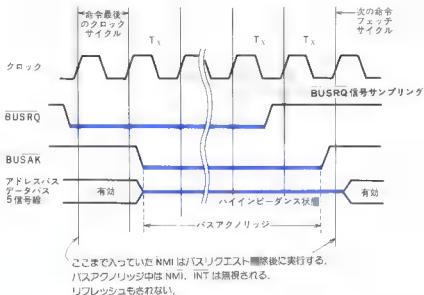
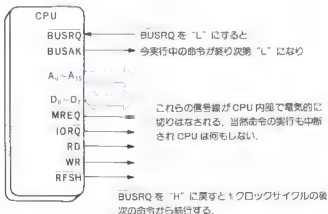
## バス要求と応答

**通**常 CPU はバスを占有して動作していますが、マルチ CPU システムで他の CPU からの要求や、その他いろいろな理由で一時的動作を中止してバスを空けたいことがあります。このようなとき**バスリクエスト (BUSRQ)** 信号を与えると、トライステートの端子をすべて高インピーダンスにし、**バスアクノリッジ (BUSAK)** 信号をアクティブにして、外部に対してバスを空け渡したことを知らせることができます。この間メモリフレッシュは行われなくなりますので、長時間この状態を続けるときは注意が必要です。

バスリクエスト (**BUSRQ**) 信号は、割り込み信号と同様に、命令の最後のクロックサイクルの立ち上がりの時点で検知されます。ここでバスリクエスト (**BUSRQ**) が“L”であると、次の命令のフェッチサイクルはなくなり、バスアクノリッジ (**BUSAK**) がアクティブになります。これ以後各クロックの立ち上がりでもバスリクエスト (**BUSRQ**) が監視され“H”に戻っていると、次のクロックサイクルから元の動作に戻ります。バスアクノリッジ (**BUSAK**) がアクティブな期間は、ノンマスカブルインタラプトとインタラプトは受け付けられなくなります。しかしバスリクエスト (**BUSRQ**) が最初に検知される時点での割り込みは受け付けられ、バスアクノリッジ (**BUSAK**) の解除後に、それぞれの動作に入ります。

バスリクエスト (**BUSRQ**) 端子を使用しないときは、 $V_{CC}$  へつないで“H”レベルにしておきます。





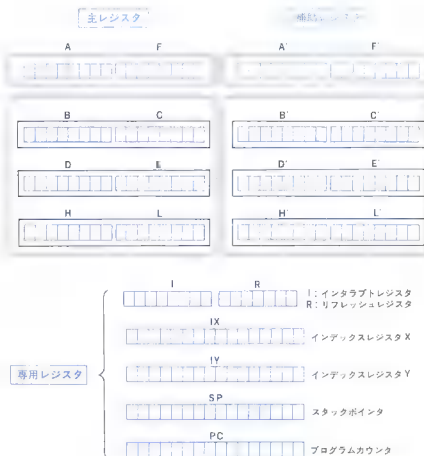
## 内部レジスタの構成

**C** PU 内部には、種々の作業に使うための一時的なメモリがたくさんあります。1ビット単独のものは、フリップフロップ (F/F) と呼ばれ、複数ビットが一連にして使われるものは、レジスタと呼ばれます。このうちでユーザに公開されるレジスタが22個あります。それぞれに個性を持っていて、うまく使い分けることが、よいプログラムを作るうえで重要です。どのレジスタをどう使うかは一概にはいえず、ケースバイケースで考えなければなりません。プログラムをいくつも作るうちに、だんだんとうまくなっていくでしょう。

8ビット単独で働くレジスタは、AレジスタとFレジスタです。Aレジスタはアキュムレータとも呼ばれ、最も重要な働きをします。Fレジスタはフラグレジスタとして各ビットに意味を持つレジスタです。8ビット単独でも、二つをつないで16ビットとしても働くレジスタは、B、C、D、E、H、Lレジスタです。ここまでの8個のレジスタは、主と補助の2組あり、内容を交換することができます。

専用レジスタとして用途が決められているものは6個あります。割り込み処理で使われるIレジスタは8ビット。メモリアドレスをカウントしているRレジスタは7ビットです。作表に便利なインデックスレジスタはIX、IYの2本あり、各16ビットです。スタックのアドレスを入れておくスタックポインタとプログラム実行アドレスをカウントするプログラムカウンタも16ビットです。

16ビットレジスタには2桁、8ビットレジスタには1桁の名称が付けられており、補助レジスタには‘(ダッシュ)’を付けて、たとえばA'と呼ぶようになっていきます。



## A, I, R, Fレジスタ

**レ**ジスタ群の中で最も多用される A レジスタは、人が手作業で計算するときのソロバンにたとえられます。加算を行なう場合を例にとると、まず足される数を A レジスタへ入れます。次に A レジスタに対して、別の足す数を加えると、A レジスタの内容が答になるわけです。このような用途のレジスタを**アキュムレータ**と呼びます。

8 ビットの加算、演算、論理和、論理積、排他的論理和、比較は、すべて A レジスタの内容に対して行なわれます。

I レジスタと R レジスタに読み書きするときは、一度 A レジスタを経由しないとできません。入力、出力も、ポートアドレスに絶対番地を指定するときは、A レジスタを経由します。

I レジスタは、割り込みモードを 2 に指定したときに、割り込み処理ルーチンへジャンプするためのジャンプ先のアドレス指定のあるアドレスの上位 8 ビットを表わします。初期値設定のために 1 回書き込むだけで、ほとんどいじる必要はないレジスタです。

R レジスタは、リセット信号でゼロになり、フェッチサイクルごとに 1 ずつ増加して、リフレッシュ用のアドレスを指定しています。プログラムで内容を変えることはほとんどありません。むしろ正確にリフレッシュさせるためには、さわるらないほうがよいのです。

F レジスタは、プログラムで内容を変えることはできません。1 命令実行ごとに、その命令により出てきた結果に特別の意味があれば、それ以後の命令実行に必要な情報を記憶します。たとえば、減算命令を実行して結果がゼロであれば、F レジスタ中の 7 番目のビット（これをゼロフラグといいます）を“1”にしてこのことを覚えておきます。次の命令が、「結果がゼロならばジャンプせよ」といった命令である場合、これをチェックして判断するようになっています。このように種々のフラグ（旗）が並んでいるレジスタが F レジスタです。



図 4-1 Aレジスタの動作

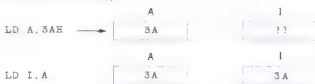


図 4-2 Iレジスタの動作

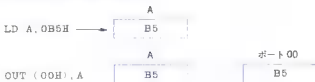


図 4-3 Fレジスタ



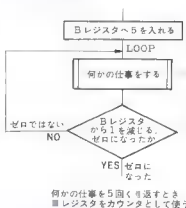
## 汎用レジスタ

**汎** 用レジスタには、H、L、B、C、D、Eレジスタの6個と、同数の補助レジスタがあります。2個のレジスタをペアにして、HL、BC、DEと表現し、16ビットレジスタとして使うこともできます。

8ビットレジスタとしての働きは、主に演算途中の数値などを一時記憶することですが、Bレジスタはループ回数を入れておき、1回に1ずつ減算し、ゼロになったら、ループから出るというカウンタの役目をさせることがあります。このためにデクリメントジャンプノンゼロ〔DJNZ〕命令がとても便利です。Cレジスタは入力〔IN〕、出力〔OUT〕命令で、ポートアドレスを指定するポインタとして使えます。

16ビットレジスタとしてペアにして使うと、HLは加算、減算でアキュムレータとして働きます。また、アドレスを入れ、間接的にアドレス指定をすることができます。

くり返し処理でアドレスを入れておくレジスタを**ポインタ**、回数を入れておくレジスタを**カウンタ**と呼びます。ブロック転送〔LDIR〕命令は、元のアドレスをHL、先のアドレスをDE、ブロックの長さ〔バイト数〕をBCに入れてから実行すると、HLの内容番地のメモリからDEの内容番地のメモリへ1バイト転送し、HLとDEに1を加え、BCから1を引きます。これをくり返しBCがゼロになると実行を終える、という命令です。HLとDEをポインタとして、BCをカウンタとして使用しています。



ラベル	オペコード	オペランド
	LD	B, 5
LOOP		
	DJNZ	LOOP

DJNZ: Bレジスタから1を引いて  
結果がゼロなら次の命令を、  
ゼロでなければLOOP番  
地の命令を実行しろ

### レジスタ

Cレジスタの内容が00H

Aレジスタの内容が5AHとすると

OUT (C), A	A 5A	C 00	ポート 00 5A
------------	---------	---------	--------------

レジスタの内容をポート 00 に出力する

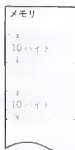
### メモリ

BCレジスタへ10を入れる	LD BC, 10
HLレジスタへ0100Hを入れる	LD HL, 0100H
DEレジスタへ0200Hを入れる	LD DE, 0200H
HLの内容を番地とするメモリから DEの内容を番地とするメモリへ 1バイト転送し HLとDEにおのおの1を加え BCから1を引け BCがゼロになるまでくり返せ	LDIR

次の番地を示す  
カウンタを減らす

0100

0200



ブロック  
転送  
そっくり  
転送し内  
容が同じ  
になる

## 補助レジスタと交換命令

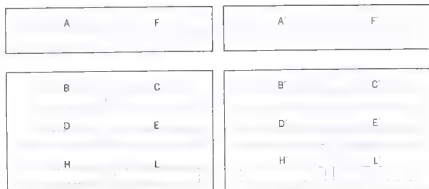
**A**、F、H、L、B、C、D、Eの各レジスタは、同じものがもう1組あります。補助レジスタと呼び、主レジスタとは区別して、“'”ダッシュを付けて表現します。補助レジスタの内容は、交換命令と呼ばれる一連の命令で、主レジスタの内容と入れ換える以外に操作することはできません。したがって、主レジスタの裏側にあって、交換命令でひっくり返すと、表（主）へ出てくると考えられます。補助レジスタとの交換命令には、AFとAF'のペアを交換する〔EX AF, AF'〕と、上記以外の汎用レジスタを交換する〔EXX〕の二つがあります。

サブルーチンや割り込み処理ルーチンの中でメインルーチンで使用中のレジスタの内容を変えたくないことがあります。このようなときは、サブルーチンに入ったところで、レジスタの内容をメモリへ待避させてから、必要なレジスタを使い、サブルーチンから出る前に待避した内容をレジスタへ戻せばよいのですが、交換命令を使えば、1命令で全レジスタを待避させることができます。特に、スピードが問題になる割り込み処理の場合は都合のよい命令です。ただし、メインルーチンで主レジスタ、サブルーチンで補助レジスタを使うことをきちんと取り決めておかないと、どちらに何が入っているのかわからなくなります。

交換命令には他にDEとHLを交換する〔EX DE, HL〕と、HL, IX, IYの各内容とスタックの内容を交換する〔EX (SP), HL〕, 〔EX (SP), IX〕, 〔EX (SP), IY〕があります。

なお、通常の転送命令の場合は、たとえば〔LD A, B〕であれば、Bレジスタの内容がAレジスタへ転送されるだけで、元のAレジスタの内容はなくなってしまいます。Bレジスタは元のまま変わりません。交換命令では双方ともこわれることなく入れ換わります。



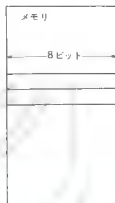


EXX

EX, EXX 以外の命令では  
主レジスタしか扱えない



EX DE, HL



n

n+1



EX (SP), HL



EX (SP), IX

EX (SP), IY

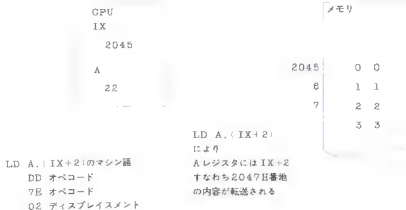
## IX, IY レジスタ

IX, IY レジスタは、インデックスレジスタと呼ばれ、同一の働きをする 16 ビットレジスタが 2 本あります。通常はここへメモリのアドレスを入れて、間接的にアドレス指定をするときに使います。表を操作するときなどに表の先頭アドレスを入れておき、1 ずつ増加させながら次々に読み出すことができます。また、先頭から何番目のデータが欲しいというようなときは、先頭からの距離をプログラムで指定することもできるレジスタです。たとえば、`[LD A, (IX+5)]` と書けば、IX レジスタに入っている数プラス 5 番地のメモリ内容を A レジスタへ入れることができます。距離は、 $-128 \sim +127$  バイトまでの範囲に限られます。距離のことをディスプレイメントと呼びます。

このレジスタの内容を間接アドレスとしてソースまたはディスティネーションに指定した命令のマシン語は、オペコードが 2 バイト構成になります。3 バイト目には、ディスプレイメントの値が入ります。オペランドがある場合は 4 バイト目に続きます。

実行のサイクルは、ディスプレイメントの演算のためのクロックサイクルが入るため、例外的に長くなります。`[LD A, (IX+5)]` では、19 クロックサイクルです。

たとえば IX レジスタに 2045H が入っていると



メモリに 3 バイトを一組とするデータ  
があるとする

{

<p>LD IX, 2040H</p> <p>LD DE, 3</p> <p>LOOP LD A, (IX)</p> <p>LD B, (IX+1)</p> <p>LD C, (IX+2)</p> <p style="margin-top: 10px;">{</p> <p>ADD IX, DE</p> <p>JP LOOP</p>	<p>最初の組の先頭番地を IX へ</p> <p>DE へ 3 を入れておく</p> <p>1 バイト目を A レジスタへ</p> <p>2 バイト目を B レジスタへ</p> <p>3 バイト目を C レジスタへ</p> <p style="margin-top: 10px;">A, B, C レジスタにある</p> <p>1 組を処理する</p> <p>次の組の先頭番地を IX へ</p>
--	---



## スタッカとスタックポインタ (SPレジスタ)

**サ**ブルーチンコール(CALL)命令を実行したり、割り込みルーチンへジャンプしたりするときに、戻り番地を自動的に記憶する機能があることは前述しました。これ以外にもプログラム中で一時的にレジスタの内容をメモリに退避したいことがよくあります。このようなときに大変便利なのがスタッカです。

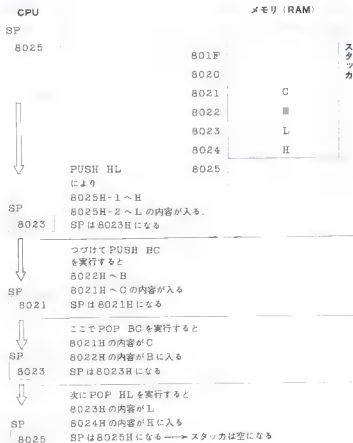
スタッカはメモリ(RAM)の一部を割り当てた記憶領域のことですが、このアドレスは、スタックポインタ(SP)と呼ばれる16ビットレジスタに入れておきます。

スタックポインタに8025Hという値がプログラムによって入れられているとしましょう。このとき(PUSH HL)を実行すると、8025H-1の8024HへHレジスタの内容が、8025H-2の8023HへLレジスタの内容が書き込まれ、スタックポインタは8023Hになっています。次に読み出すときは(POP HL)を実行すると、8023Hの内容をLレジスタに、8023H+1の内容をHレジスタに転送し、スタックポインタは8025Hになります。

スタッカに次々におさめられた値は、後から入れたほうが先に読み出されますので、First In Last Out (FILO) タイプのスタッカと呼ばれます。サブルーチンや割り込みルーチンに入るときは自動的に(PUSH PC)\*に相当する命令が実行され、戻るときは(POP PC)\*に相当する命令が実行されていることになります。したがって、サブルーチンの中から次のサブルーチンをコールするときも、(これをサブルーチンのネスティングといいますか)うまく働いてくれます。ただし、スタッカとして割り当てた領域を超えるネスティングはできません。未完成のプログラムで誤って次々にサブルーチンコールをすると、このようなことが起こることがあります。プログラム設計の際に、スタッカの大きさを決定するときは、慎重な判断を要します。またプログラムの始まりでは必ずスタックポインタのイニシャライズ(初期値設定)を行なってください。(PUSH)、(POP)できるベアレジスタはAF, BC, DE, HL, IX, IYです。

\* この命令は、プログラム上では使えない。

# 47 スタックとスタックポインタ (SP レジスタ)



プログラム中ではいちいち SP の値を気にしなくてもよい。

AA    PUSH HL    ただし、PUSH と POP は対にして使わなければならない  
       PUSH BC    (意図した実則的な場合もある)

この間で H、L、B、C の内容は変わっても  
 AA での内容と BB での内容は同じになっている。

POP BC    CALL では PUSH PC  
 POP HL    RET では POP PC が実行されているのと同じなので、  
 BB        かならず対にして使う

## 転送命令

### 最

も基本的な転送命令は、ロード命令〔LD a, b〕です。オペランドにはソースとディスティネーションの指定をします。ニモニックはZ-80ではすべて〔LD〕で統一されています。8ビットの転送では、ソースにはレジスタ名、ベアレジスタによる間接アドレス、インデックスレジスタによるディスプレイメント、直接アドレス値および定数が指定できます。ディスティネーションには、ソースで利用できるうちの定数以外を指定できます。ただし、I, RレジスタはAレジスタとのやりとりしかできないなど、ソースとディスティネーションの組み合わせには限定があります。

16ビット転送では、ベアレジスタとやりとりのできる組み合わせは、16ビット定数、直接アドレス値だけで、ベアレジスタ間のやりとりは、HL, IX, IYレジスタからスタックポインタ〔SP〕への転送だけになります。

アセンブリ語で記述する場合、アドレス値や定数は、10進数、16進数で書いてもよいのですが、ラベル名で記述することもできます。マシン語に直したときは、アセンブリリスト上は、16進数に統一されます。16ビット定数やアドレス値のときは、オペコードの次に下位8ビット、その次に上位8ビットがくるように並びます。

レジスタや定数に〔 〕かっこを付けたオペランドは、メモリへまたはメモリからの転送命令を意味します。かっこ内の数値やラベルは、メモリのアドレスを指しています。16ビット転送の場合は、このアドレスが下位8ビット、次が上位8ビットの転送アドレスになり、ベアレジスタとの転送では、たとえばHLならば、Hレジスタが上位、Lレジスタが下位に対応します。

〔PUSH〕、〔POP〕も16ビット転送命令ですが、スタックポインタの項で述べました。

## 8 ビットロード

ディスティネーション (受け側)		ソ ー ス (送 り 側)		
		レ ジ ス タ	メ モ リ	定 数
レ ジ ス タ	A	A, B, C, D, E, H, L, I, R	(HL), (BC), (DE), (IX+d), (IY+d), (定数)	定数
	B	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
	C			
	D			
	E			
	H			
	L			
	I R	A		
メ モ リ	(HL)	A, B, C, D, E, H, L		定数
	(BC) (DE)	A		—
	(IX+d) (IY+d)	A, B, C, D, E, H, L	—	定数
	(定数)	A		

アドレス

Aレジスタはすべての  
転送ができる

メモリーメモリの転送はできない

■ビット定数

## 16 ビットロード

ディスティネーション (受け側)		ソ ー ス (送 り 側)		
		レ ジ ス タ	メ モ リ	定 数
レ ジ ス タ	BC DE HL	HL, IX, IY	(定数)	定 数
	SP			
	IX IY			
メモリ	(定数)	BC, DE, HL, SP, IX, IY	—	—

PUSH, POPのできるレジスタ  
AF, BC, DE, HL, IX, IY

LD (nn), HL や  
LD HL, (nn) の  
場合  
nn  $\wedge$  L (下位) nn  
nn+1  $\wedge$  H (上位) nn+1  
が対応する

メモリー	
nn	L
nn+1	H

## 算術演算命令

**レ**ジスタやメモリ内の8ビットないし16ビットのビットパターンを2進数とみなして、数学的な演算をする命令があります。8ビットの演算は、アド[ADD]、サブトラクト[SUB]、アドウィズキャリ[ADC]、サブトラクトウィズキャリ[SBC]の4命令があります。Aレジスタに対して、他のレジスタかメモリの内容を加減算します。この演算のとき桁あふれ、または桁借りをする命令と無視する命令に分かれます。あふれた桁はFレジスタの1ビット目のキャリフラグ(Cフラグ)があてられます。

コンペア[CP]命令は、Aレジスタからオペランドの内容を減算しますが、Aレジスタの内容は変わらず、Fレジスタが減算のときと同じに変化し、あとの命令で、結果がゼロであったかチェックできます。ゼロであったときは、Aレジスタの内容とオペランドの内容が同じであると判断できます。

インクリメント[INC]命令は、オペランドに1を加える命令です。デクリメント[DEC]は、オペランドから1を引く命令です。メモリ内のデータ列を1バイトずつ順次操作する場合やくり返し行なう仕事の回数をカウントするときに使います。

ニグイト[NEG]は、Aレジスタの内容をゼロから引いてAレジスタに入れる命令です。すなわち、Aレジスタの内容の符号(プラス、マイナス)を反転させるとき主に使います。

16ビットの演算は、HL、IX、IYレジスタの内容に対して行なう加減算で、桁あふれを考えに入れない演算は加算だけです。インクリメント[INC]とデクリメント[DEC]は、8ビットと同様です。メモリアドレスの操作に使います。

算術演算命令でのFレジスタの働きは重要で、多桁の演算では、キャリフラグの働きに注意する必要があります。また10進補正[DAA]命令は、Fレジスタの結果を参照して補正のし方を決定します。



	命令	ディスティネーション 被演算数・答	ソ                      ス			操 作 S: ソース Cy: キャリ
			レジスタ	メモリ	定数	
8 ビ ッ ト	ADD	A				$A + s \rightarrow A$
	ADC	A		(HL)		$A + s + Cy \rightarrow A$
	SUB	A*	A, B, C, D	(IX+d)	定数	$A - s \rightarrow A$
	SBC	A	E, H, L	(IY+d)		$A - s - Cy \rightarrow A$
	CP	A*				$A - s$ A は不変
	INC	ソースと同じ*	A, B, C, D	(HL) (IX+d)		$s + 1 \rightarrow s$
	DEC	ソースと同じ*	E, H, L	(IY+d)		$s - 1 \rightarrow s$
	DAA	A*	A*			A を補正
	NEG	A*				$0 - A \rightarrow A$
16 ビ ッ ト	ADD	HL	BC, DE HL, SP			$HL + s \rightarrow HL$
	ADD	IX	BC, DE SP, IX			$IX + s \rightarrow IX$
	ADD	IY	BC, DE SP, IY			$IY + s \rightarrow IY$
	ADC	HL	BC, DE			$HL + s + Cy \rightarrow HL$
	SBC	HL	HL, SP			$HL - s - Cy \rightarrow HL$
	INC	ソースと同じ*	BC, DE, HL			$s + 1 \rightarrow s$
	DEC	ソースと同じ*	SP, IX, IY			$s - 1 \rightarrow s$

\*印はアセンブリ語ではディスティネーションはオペランドに書かない。(例) SUB C INC A  
DAA NEG ではソースも書かない。

ADC では C フラグ (キャリ) もここへ加える。  
すなわち、さらに下の桁の演算がこの前に行な  
われているとすると、そのときの桁上りが含ま  
れる

```

10111001
+ )01010101
-----
100001110

```

桁上り C フラグ (キャリ) へ入る

- SBC は、桁借りが C フラグに入っているのをこの演算を含めての演算をする。
- SUB は、前の演算で桁借りがなかったものとして C フラグは無視してしまう。

## 論理演算命令

**論** 理演算命令は、すべて8ビット単位に、Aレジスタに対して行われます。各ビットごとの論理積、論理和、排他的論理和、否定をとります。

アンド〔AND〕命令は論理積です。Aレジスタとオペランドの内容の双方共に“1”のビットだけ“1”が残り、他は“0”になります。Aレジスタのある特定のビットだけ残したいときは、ここを“1”、他を“0”としたビットパターンをオペランドで指定して、アンドをとれば、不要なビットは必ず“0”になり、必要なビットだけが“1”であれば“1”、“0”であれば“0”として残ります。この方法を**マスク**といいます。Z-80ではビット操作命令があるのであまり使いません。

オア〔OR〕命令は論理和です。Aレジスタとオペランドの内容のいずれかが“1”のビットを“1”として、双方共に“0”のビットを“0”にします。16ビットの演算のデクリメント〔DEC〕命令でHLレジスタの内容を1ずつ減らしてゼロになってもフラグは変化しません。そこで、このチェックを行なうときは、HレジスタをAレジスタに移して、Lレジスタとのオア〔OR〕をとりますと、H、L、共に全ビットが“0”、すなわちHLレジスタとしてゼロになっているときだけ結果がゼロになり、フラグレジスタのゼロフラグがこのことを示します。また、特定のビットを前の内容にかかわらず“1”にしたいときには、このビットが“1”、他が“0”のビットパターンとオア〔OR〕をとれば、“1”のところは“1”になり、“0”のところは前のまま残ります。

エクスクルーシブオア〔XOR〕命令は排他的論理和です。Aレジスタとオペランドの内容が、ビット単位に同じところを“0”、異なるところを“1”にします。AレジスタとAレジスタのエクスクルーシブオア〔XOR〕をとるとAレジスタは必ずゼロになります。なおキャリフラグだけを“0”にするときは、〔AND A〕が〔OR A〕を使います。

コンプリメント〔CPL〕命令はビットパターンをすべて反転させます。すなわち、“1”のビットは“0”に、“0”のビットは“1”にする論理否定です。

命令	ディスティネーション	ソ                      ス		
		レ                      ジ                      ス                      タ	メ                      モ                      リ	定 数
AND OR XOR	A	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	定数
CPL	A	A		

アセンブリ言語ではディスティネーションはオペランドに書かない。

CPL ではソースも書かない。

## AND

(例) Aレジスタ 0 1 0 0 1 0 1 1

ソース 1 1 0 1 0 0 0 1

Aレジスタ  
(結果) 0 1 0 0 0 0 0 1

a	b	結果
0	0	0
0	1	0
1	0	0
1	1	1

## OR

(例) Aレジスタ 0 1 0 0 1 0 1 1

ソース 1 1 0 1 0 0 0 1

Aレジスタ  
(結果) 1 1 0 1 1 0 1 1

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	1

## XOR

(例) Aレジスタ 0 1 0 0 1 0 1 1

ソース 1 1 0 1 0 0 0 1

Aレジスタ  
(結果) 1 0 0 1 1 0 1 0

a	b	結果
0	0	0
0	1	1
1	0	1
1	1	0

## CPL

(例) Aレジスタ 0 1 0 0 1 0 1 1

Aレジスタ  
(結果) 1 0 1 1 0 1 0 0

前	結 果
0	1
1	0

## ビット操作命令

**レ**ジスタ、メモリ内のどこの1ビットでも、ビット操作命令で“0”か“1”にすることができ、また“0”か“1”かの判定をすることができます。コントロール的な応用にはよく使う命令で、Z-80の特徴の一つといえます。

セット (SET) 命令は、ビットを“1”にする命令で、オペランドにはビットの位置とレジスタ名か、メモリのアドレスを示すレジスタ HL, IX, IY のいずれかを指定します。ビットの位置は、2 の  $n$  乗の  $n$  で表わしますので、右端が 0、左端が 7 となります。

リセット (RES) 命令は、ビットを“0”にする命令です。オペランドはセット命令と同じです。

ビット (BIT) 命令は、指定のビットを調べ、“0”であれば F レジスタのゼロフラグ (Z フラグ) を“1”にして、“0”であったことを記憶しておきます。後に書かれた条件付きジャンプ命令で、“0”のときと“1”のときの飛び先番地を別々に指定してあれば、条件に応じた手順で仕事を進めます。オペランドはセット命令と同じです。

23A6H 番地のメモリの右から 3 番目のビットを“1”にするときは、HL レジスタに 23A6H をロード命令で書き込んでから (SET 2, (HL)) を実行させ、“0”にするときは (RES 2, (HL))、内容を判定するには (BIT 2, (HL)) となります。

命 令	ディスティネーション	ソ ー ス		
		レ ジ ス タ	メ モ リ	ビット
SET RES BIT	ソースと同じ	A, B, C, D, E, H, L	(HL), (IX+d), (IY+d)	0-7

アセンブリ言語ではディスティネーションはオペランドに書かない。

HL

メ モ リ

のとき

SET 6, (HL)

7 6 5 4 3 2 1 0      ビット番号

を実行すると

0326    1    .    .    .    .    .    .

メモリの

0326H 番地の第6ビット

が "1" になる

SET    "1" にする

RES    "0" にする

BIT    指定のビットが "0" か "1" によって

Z フラグの状態を

"0" なら Z, ゼロ

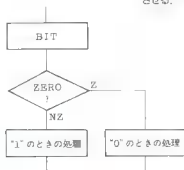
"1" なら NZ (ノンゼロ)

を示すように設定する。

この場合、後にある判定命令・ジャンプ命令

で Z フラグの状態を判定し以後の処理を分岐

させる。



BIT 0, A    Aレジスタの0ビット

JP    Z, ZRUTIN

ゼロなら ZRUTIN

番地のプログラムへ

分岐

ゼロでなければここからの

プログラムを実行

## ローテート、シフト命令

**□** ローテート命令は、ビットパターンを右か左に一つだけずらす命令です。はみ出したビットは最後に回り込んでつながります。このときキャリフラグを含めて回転するか、含めずに回転し、回り込んだビットと同じものをキャリフラグへ入れるかによって大別されます。ローテートライト (RR), ローテートレフト (RL), ローテートライトサーキュラ (RRC), ローテートレフトサーキュラ (RLC) があり、特に A レジスタに対する命令は、8080 系の CPU と互換性を持たせるために、1 バイト命令が別に用意されています。

ローテートライト (レフト) デジット (RRD), (RLD) 命令は、HL レジスタの内容で示されるメモリと A レジスタの下 4 ビットとの間で 4 ビット (1 デジット) を一まとめにして回転させる命令で、2 進法 10 進数の多桁演算によく使われるものです。

シフト命令は、はみ出したビットを切り捨て、最後には“0”または左端ビットの値 (符号ビット) が入ります。シフトライトアリスメチック (算術的右シフト) (SRA), シフトレフトアリスメチック (SLA), シフトライトロジカル (SRL) の三つです。算術的シフトは、ビットパターンを 2 進数とみなしたときシフトしても符号が変わらないよう配慮されているシフトです。左シフトは算術的 (アリスメチック) も論理的 (ロジカル) も同じですから一つしかありません。左に一つシフトすると、数値は 2 を掛けた値になり、右にシフトすると 2 で割った値になります。

ビットパターンを 2 進数扱いでかけ算、割り算をするときは、シフトとローテートを使って高速のルーチンを組むことができます。

命令	ソース (デスティネーション)		作
	レジスタ	メモリ	
RLCA	A*	-	
RLC	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RLA	A*	-	
RL	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RRC	A*	-	
RRC	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RRA	A*	-	
RR	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
RLD	A+(HL)*	(HL)	
RRD			
SLA	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	
SRA			
SRL	A, B, C, D E, H, L	(HL) (IX+d) (IY+d)	

\*印はアセンブリ言語ではオペランドに記述しない

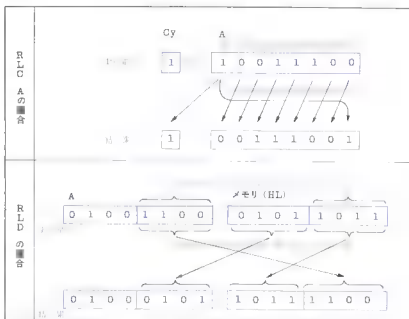
Cy は F レジスタの C フラグ (キャリ)

- SRA の 7 番ビットは変わらない。
- シフトでは Cy の元の値は失われる。
- RRD, RLD では移動する 4 ビット単位の内容は変わらない。

Cy A

1 10011100 のときの結果は

命 令	Cy	A
RLC A	1	00111001
RL A	1	00111001
RRC A	0	01001110
RR A	0	11001110
SLA A	1	00111000
SRA A	0	11001110
SRL A	0	01001110





## シフト、ローテートを使ったプログラム例

(16ビットの掛け算サブルーチン)

メモリに図のように a と b の値が2進数で入っている。

$$a \times b = c$$

の演算をし、答 c を32ビットとして、メモリに格納する。

メモリのアドレス n は IX レジスタに入れているとする。

MPX	LD	C, (IX+0)	n	メモリ	
	LD	A, (IX+1)		a	(下 位)
	LD	E, (IX+2)		a	(上 位)
	LD	D, (IX+3)		b	(下 位)
	LD	B, 16		b	(上 位)
	LD	HL, 0		c	(最下位)
	EXX			c	(下 位)
	LD	HL, 0		c	(上 位)
	LD	DE, 0		c	(最上位)
	EXX				
LOOP	SRL	A			
	RR	C			
	JR	NC, JNDT			
	ADD	HL, DE			
	EXX				
	ADC	HL, DE			
	EXX				
JNDT	SLA	E			
	RL	D			
	EXX				
	RL	E			
	RL	D			
	EXX				
	DJNZ	LOOP			
	LD	(IX+4), L			
	LD	(IX+5), H			
	EXX				
	LD	(IX+6), L			
	LD	(IX+7), H			
	RET				

## ブロック転送、ブロックサーチ、 ブロック入出力命令

× モリ内の複数バイトのブロックを別の番地へ移し変える場合、1バイトずつCPU内のレジスタへ読み込み、別の番地へ書き込む操作を必要バイト数になるまでくり返します。また、メモリ内のブロックの中から指定のビットパターンと同じパターンを持つ1バイトを探し出すときも、1バイトずつAレジスタへ読み込んでコンペア〔CP〕命令を実行し、一致するまでくり返します。入出力のときもメモリブロックの内容をポートへ次々に出力したり、次々にポートから読んだデータをメモリへ並べたりするときにはくり返しのプログラムを組まなければなりません。これを一つの命令で置き換えられるのが、この命令群です。

**ブロック転送命令**は、HLレジスタの元のアドレス、DEレジスタに転送先のアドレス、BCレジスタに転送バイト数をあらかじめ書いておき、次に転送命令を実行させますと、HLとDEレジスタの内容を1ずつ進め、BCレジスタの内容を1ずつ減らしていきます。

**ブロックサーチ命令**は、HLレジスタにアドレス、BCレジスタにブロックのバイト数を、比較すべきビットパターンをAレジスタに書いておきます。一致するとFレジスタのゼロフラグが、BCレジスタがゼロになる（一致するものがない）とオーバフローフラグ（Vフラグ）がリセットされますので、次の命令でフラグ判定をしなければなりません。

**ブロック入出力命令**は、HLレジスタにデータ格納のメモリアドレス、CレジスタにIOポートアドレス、Bレジスタにバイト数を書いてから実行させます。

これらの命令には、メモリブロックを先頭から扱うインクリメントグループと後から扱うディクリメントグループがあります。また、全データの操作が終わる（サーチでは一致した場合も含む）までプログラムカウンタが変わらないで、自動的にループするリピートの機能を持つ命令もあります。転送、サーチ、入出力しながら、何かのデータ操作がない場合は、リピートは便利です。1バイト扱うごとにフェッチサイクルから始まります。

# 53 ブロック転送、ブロックサーチ、ブロック入出力命令

(HL←HL+1の意味は、HLの内容に1を加えるということ)

命 令	動 作
LDI	HLの内容を番地とするメモリから HL←HL+1 BC←BC-1 1バイトだけ転送して終る
LDIR	DE←DE+1 BC=0まで繰り返す
LDD	DEの内容を番地とするメモリへ転送する HL←HL-1 BC←BC-1 1バイトだけ転送して終る
LDDR	DE←DE-1 BC=0までくり返す

BC = 0のときはVフラグがセットされる

## ブロックサーチ

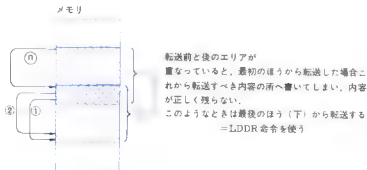
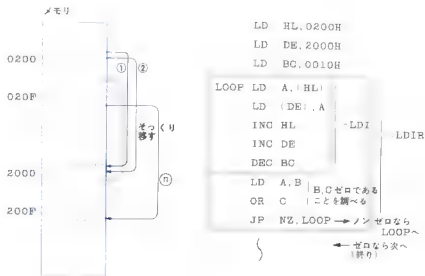
命 令	動 作
CPI	HLの内容を番地とするメモリ A←(HL)=0…等しい HL←HL+1 1バイトだけ調べて終る
CPIR	A←(HL)≠0…等しくない BC←BC-1 BC=0か A←(HL)=0まで
CPD	等しいときZフラグをセット (ゼロの状態) HL←HL-1 1バイトだけ調べて終る
CPDR	BC←BC-1 BC=0か A←(HL)=0まで

BC = 0のときはVフラグ、A (HL) = 0のときはZフラグがセットされる

## ブロック入出力

命 令	動 作	注
INI	Cの内容を番地とするポートからHLの内容を番地とするメモリへ読み込む HL←HL+1 1バイト入力して終る	Cは変わらない カウンタはBだけ (256まで)
INIR	B←B-1 B=0までくり返す	
IND	HL←HL-1 1バイト入力して終る	
INDR	B←B-1 B=0までくり返す	
OUTI	HLの内容を番地とするメモリの内容をCの内容を番地とするポートへ出力する HL←HL+1 1バイト出力して終る	
OTIR	B←B-1 B=0までくり返す	
OUTD	HL←HL-1 1バイト出力して終る	
OTDR	B←B-1 B=0までくり返す	

B = 0のときはZフラグがセットされる



## ブロック転送命令を使ったプログラム例

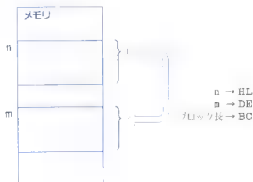
(ブロック転送サブルーチン)

HL レジスタに転送前アドレス、DE レジスタに転送後アドレス、BC レジスタに転送するブロックのバイト数が入っているとすると、転送前と転送後のエリアが重なっている場合を想定して、正しく転送できるように〔LDIR〕と〔LDDR〕を使い分ける。

```

MVE  PUSH  HL      ; HL を退避
      AND   A       ; キャリをゼロに
      SBC   HL, DE   ; キャリを変化
      POP   HL      ; HL を元に戻す
      JR    NC, IRR  ; 上から転送
      DEC   BC       ; BC-1
      ADD   HL, BC    ; HL を変更
      EX    DE, HL   ; DE を変更
      ADD   HL, BC    ; DE を変更
      EX    DE, HL   ; DE, HL
      INC   BC       ; BC+1
      LDDR  ; 下から転送
      RET
IRR   LDIR  ; 上から転送
      RET

```



## ジャンプ命令

× モリに入れられたプログラムはゼロ番地から順次実行しますが、順序を入れ換えたり、条件によって別の番地のプログラムへ分岐したりするときに使います。

**ジャンプ〔JP〕命令**は、条件を伴わない無条件ジャンプと、条件付きジャンプ命令があり、飛び先番地は絶対番地が付けられます。

条件付きジャンプ命令は、フラグレジスタの内容によって、ジャンプするかジャンプせずに次の命令へいくかの決定をします。この条件には、ゼロフラグ、キャリフラグ、パリティフラグ、サインフラグがそれぞれ“0”である、または“1”であるとき、ジャンプせよというものです。条件に合わないときは、そのジャンプ命令がないのと同様に次の命令を実行します。

**ジャンプリラティブ〔JR〕命令**は、ジャンプ先のアドレスを絶対値で持つのではなく、現在のプログラムカウンタ（自分自身のアドレス）からの隔たりで持っています。数値としては -128 から +127 バイトで、0 のときは次の命令の先頭番地です。アセンブリ言語で書くときは、飛び先番地のラベルを指定すれば、自動的に計算してくれます。ジャンプリラティブ〔JR〕命令は 2 バイト命令ですが、ジャンプ〔JP〕命令（3 バイト）より実行時間がかかります。また条件判定は、ゼロフラグとキャリフラグについてのみしかできません。しかしジャンプ先までの隔たりが変わらなければジャンプ先がどこの番地になってもマシン語は同じになりますので、小さなプログラムモジュール内で使った場合、モジュールの配置は自由（リロケータブル）となります。ディクリメントジャンプノンゼロ〔DJNZ〕命令は、変わった命令で、B レジスタから 1 を引き、ゼロでなければジャンプリラティブと同じ方式で示された番地へジャンプするというものです。くり返しに入る前に B レジスタへ必要回数を書き、くり返し処理の終わりにこの命令をおけば、終了しなければもう 1 回くり返しの先頭へジャンプ、終了すれば抜け出て次へ、と分岐させることができます。回数は B レジスタが 8 ビットですから 256 回（B レジスタにゼロを入れておくと 256 回で終わる）までです。

## 無条件ジャンプ

命 令	オ ペ ラ ン ド
JP nn	飛び先のメモリにつけられた番地をそのままオペランドにつける(絶対番地) (例) JP 0258H ⇒ マシン語では C3 58 02 アセンブリ言語ではラベル名を書いてよい <sup>T位 上位</sup> オペランドには(HL)(IX)(IY)を使用してもよい。
JR nn	飛び先のメモリの番地までのディスプレイメントをオペランドにつける(相対番地) (例) JR L02 ⇒ L02が0255H番地で次の命令が0250H番地 とすると、マシン語では18 05

## 条件付きジャンプ 条件が合えばジャンプする、合わなければ次の命令へ

命 令		条 件
JP	JR	
JP Z, nn	JR Z, nn	Zフラグを調べゼロの状態(Zフラグ=1)ならジャンプ
JP NZ, nn	JR NZ, nn	Zフラグを調べノンゼロの状態(Zフラグ=0)ならジャンプ
JP C, nn	JR C, nn	Cフラグを調べキャリ有り(Cフラグ=1)ならジャンプ
JP NC, nn	JR NC, nn	Cフラグを調べキャリなし(Cフラグ=0)ならジャンプ
JP PE, nn	-	P/Vフラグを調べイーブン(偶数)(P/Vフラグ=1)ならジャンプ
JP PO, nn	-	P/Vフラグを調べオッド(奇数)(P/Vフラグ=0)ならジャンプ
JP M, nn	-	Sフラグを調べマイナス(Sフラグ=1)ならジャンプ
JP P, nn	-	Sフラグを調べプラス(Sフラグ=0)ならジャンプ
-	DJNZ nn	B←B-1 B≠0ならジャンプ

注 JRのオペランドはアセンブラによってはラベル名ではなく、相対値を書くものもある。

# 54 ジャンプ命令

;	
LD	B, 10H
LD	HL, 2000H
LD	A, 00H
LO1	ADD A, (HL)
	INC HL
	DJNZ LO1
- Bレジスタから1を減じゼロなら次の命令を実行する、 ゼロでなければLO1番地へジャンプ	
	JP KEYR
- KEYRとラベルをつけた番地へジャンプ (この次はKEYR番地の命令を実行する)	
CRTR	LD A, 30H
	LD C, 00H
- ここへは他にあるジャンプ命令等でジャンプしてくる。	
;	
JR	LO2
- LO2のラベルがつけられた命令の先頭番地。次の命令の先頭番地の値をマシン語のオペランドに持つジャンプ命令(2バイト構成の命令)、LO2とラベルをつけた番地へジャンプ	
KEYR	IN A, (08H)
CP	0
	JP Z, KEYO
- 前の命令でAレジスタと0を比較してゼロ(一致)ならKEYO番地へジャンプ	
	CP 1
- ゼロでなければこの命令を実行する	
;	
KEYO	LD B, 20H
LO2	LD HL, 003BH
;	



## よいプログラムとは

### 1. 実行速度が速いこと

用途によっては、速度が問題にならない場合もあるが、まずほとんどの場合、全システムの効率に大きく影響する。

### 2. メモリをくわないこと

プログラム自体の長さは、無駄を省いた最小限でなければならない。

実行速度を上げるためにも重要である。また、専用システムではメモリ数を減らし、部品点数を引き下げることにより得られるメリットは多い。ただし、以下のことを犠牲にしないよう心がけなければならない。

データ納用メモリは、情報処理（事務計算）のような用途で多用されるが、フロッピディスクなどの外部記憶を有効に利用し、システム効率を上げるような設計が望ましい。

### 3. テバグ（修正）しやすいこと

いかに有能なプログラマでも、できあがったプログラムが1回で動作するのはまれである。また、できあがってから何年もたってからバグ（誤り）が発見されることがある。すみやかに対処できるよう配慮しなければならない。次項とも共通するが、コメントを豊富に、機能をモジュール化し分散させること、フラグの使用は最小限にする。サブルーチンからジャンプでメインルーチンへ戻るなど論外である。

### 4. 改造しやすいこと

新しくプログラムを作るとき、前に作ったプログラムの改造で対応できれば、こんな楽なことはない。設計変更されそうな点を予測すること。

### 5. 速く作ること

プログラムの原価は、ほとんど人件費につく。速く、しかも後で応用のきくプログラムを作ることが、プログラマの使命である。

## コール、リスタート、リターン命令 (サブルーチン)

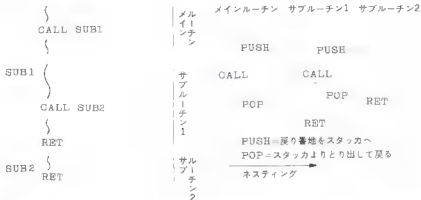
**コール (CALL) 命令**は、ジャンプ命令と似ていますが、ジャンプするとき次の命令の番地をスタックへ自動的にプッシュ (PUSH) します。そしてリターン (RET) 命令があると、スタックからポップ (POP) した値をプログラムカウンタへ入れて元の流れへ戻ります。コールで呼び出される番地からリターン命令までをサブルーチンと呼び、プログラムの中のどこからでも呼び出して実行させることができます。コール命令は、2 バイトのジャンプ先の絶対番地を持った 3 バイト命令です。コール命令にも、ジャンプ命令と同じ条件を付けることができます。条件を満たしていないときは、コール命令がないのと同様に、ジャンプせずに次の命令を実行します。

**リスタート (RST) 命令**は、特定の 8 個の番地へのコール命令に相当します。1 バイト命令なので、多用されるサブルーチンをコールするときに便利な命令です。

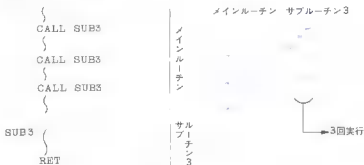
リターン命令であっても、ノンマスクابل割り込み (NMI) からのリターンはリターンフロムノンマスクابلインタラプト (RETN) を使います。他の割り込みに対する割り込み禁止状態を解除するためです。また、割り込み (INT) からのリターンは、Z-80 のペリフェラルのデージェーチェーンによる優先順位決定機能を使うときは、リターンフロムインタラプト (RETI) 命令を使って、ペリフェラルに対して割り込み処理の完了を知らせてやる必要があります。これは割り込み処理ルーチンの中で、サブルーチンコールを行なう場合もあり、サブルーチンからのリターンで割り込み処理完了になってしまわないようにするためです。

サブルーチンの使い方は、プログラムの良否にかかわる重要なテクニックです。単に同じ手続きをあちこちで何回も使うから、記述の手間を省くというだけでなく、プログラムの機能のブロック化、あるいは単機能モジュールのブラックボックス化に役立ちます。デバッグ済みのサブルーチンは内容を知らなくても使い方をだけを知っておけば、いつでも呼び出せます。モジュールとしてたくさん作っておけば、次からはモジュールの組み合わせを変えるだけで、新しいプログラムができあがるわけです。

メインルーチンからサブルーチン1をコールし、さらにサブルーチン2をコールする



メインルーチンのあちこちから1つのサブルーチン3をコールする



命 令	オ ペ ラ ン ド	
CALL nn	nnは絶対番地	無条件コール
CALL cc, nn	nnは絶対番地 ccは条件	条件付コール 前項のJPと同じ条件
RET	オペランドはない	無条件リターン
RET cc	ccは条件	条件付リターン 前項のJPと同じ条件
RST n	nは 00H 08H 10H 18H 20H 28H 30H 38H のいずれかを選ぶ n=08Hとすると、0008Hへのコール命令 (CALL 0008H) と同じ働きをする。ただし RST nは 1バイト命令	

## F レジスタとフラグ変化

**F** レジスタは六つのフラグにより構成されます。フラグはそれぞれが意味を持ち、演算命令やローテート命令、入出力命令などの実行結果によって変化します。

**キャリフラグ (C フラグ)** は、最上位のビットからの桁上がり、桁借りにより、あるいはローテート、シフト命令で変化します。

**ゼロフラグ (Z フラグ)** は、実行結果がゼロになったとき“1”にセットされます。

**パリティ/オーバーフローフラグ (P/V フラグ)** は、パリティ (ビットパターンの“1”の数が奇数か偶数か) と、符号付き演算のオーバーフローを兼ねています。パリティは、奇数なら“0”、偶数なら“1”、オーバーフローは、オーバーフローして本来正の数になるべき結果が負の数になってしまったとき“1”、正しい結果のとき“0”になります。

**サインフラグ (S フラグ)** は、符号付き演算の符号ビット (左端ビット) と同じになります。負の数であれば“1”、正の数であれば“0”です。

**ハーフキャリフラグ (H フラグ)** は、下位4ビットに対するキャリフラグです。これらのフラグのうち、サブトラクトフラグとハーフキャリフラグは、ジャンプ命令では判定できませんが、2進法10進数を扱うデシマルアジャストアキュムレータ [DAA] 命令を実行するとき参照されます。

フラグは命令により変化する、変化しない、不定になる、各場合があります。

**コンプリメントキャリフラグ [CCF] 命令**は、他のレジスタに変化を与えずにキャリフラグのビットパターンを反転させる命令です。セットキャリフラグ [SCF] 命令はキャリフラグを“1”にする命令です。なお“0”にしたいときは [AND A] か [OR A] を使います。

どの命令を実行したときどのフラグが変化するかは、巻末のフラグ変化表と命令表を参照してください。

## Fレジスタ



## 2 進化 10 進数と 10 進補正命令

**C** PU 内の算術演算は、メモリやレジスタに記憶されたビットパターンを 2 進数として演算します。8 ビットレジスタでは 0~255 まで、256 以上を表わすと桁上げをしますが、レジスタの内容は 0 と 256 が同じパターンになります。多くの桁を扱う場合は、桁上げ操作や入出力を簡単にするため、2 進化 10 進数(バイナリコードデッドデシマル=BCD)を用います。

4 ビットのパターンは 16 種類 0~F までありますが、このうち 0~9 までを有効とし、A~F については上位桁の 4 ビットへ 1 くり上げ 0~5 にします。このようにすると 4 ビット=1 桁の 10 進数表現ができます。10 桁の 10 進数を使いたいときはメモリに 5 バイトのエリアを確保し、2 桁ずつ演算するのです。

しかし、演算は BCD で表現されていても、2 進数で計算しますので、結果として A~F が出現してしまいます。ここでデシマルアジャストアキュムレータ(DAA)命令を実行すると、いまの計算が加算か減算か、ハーフキャリは出ているか、などフラグレジスタの状態によって必要な桁上げ、桁借り処理をして、BCD に戻してくれます。BCD はデータエリアが多少大きくなりますが、入出力はほとんどの場合 10 進数が要求されますので、変換の手間が省け、多桁演算がやりやすいなど特徴が多く、よく使われる手法です。

ビットパターン	名前	BCD表現
0 0 0 0	0	ここまでを 10進数に対応 させる 1010は 上の位へ桁上げし 0000に戻る
0 0 0 1	1	
0 0 1 0	2	
0 0 1 1	3	
0 1 0 0	4	
0 1 0 1	5	
0 1 1 0	6	
0 1 1 1	7	
1 0 0 0	8	
1 0 0 1	9	
1 0 1 0	A	使わない
1 0 1 1	B	
1 1 0 0	C	
1 1 0 1	D	
1 1 1 0	E	
1 1 1 1	F	

8+6を例にとって

加算すると

10進数では

8+6=14であるが

2桁のBCDでは

0000 1000 = 08

+ 0000 0110 = 06

0000 1110 = 0E

となり、下4ビットは使わない値になってしまう。

ここでDAA命令を実行すると

0000 1110

0001 0100

1 4

になおしてくれる

この補正のし方は加減算、

桁あふれ等により異なるが、

レジスタの働きで補正できることはない

1001 0110

9 6  
↑ ↑  
10の位 1の位

1桁の10進数

(4ビットを1デジット(桁)と呼ぶのは  
ここからきている)

2進数

1001 0110 は

16進数 になおすと(名前と呼ぶと)

96である、これを

10進数になおすと

$9 \times 16^1 + 6 \times 16^0 = 150$  である

1001 0110 は

16進読み にすると

96 (きゅうろく)

10進数にしても

96 (きゅうじゅうろく) である

コンピュータの中で数値を表わすには

1. 2進数として扱う
2. BCDとして扱う
3. ■■■表現をとり入れる

等のやり方がある

- 1. は入出力のたびに10進数との変換が必要。プログラムは簡単でスピードも早い。桁数が自由にならない。
- 2. は桁数が自由になり、入出力、プログラムは簡単。メモリにむだが多い。
- 3. は、桁数が多い場合に有効。プログラムは複雑で演算時間がかかる。

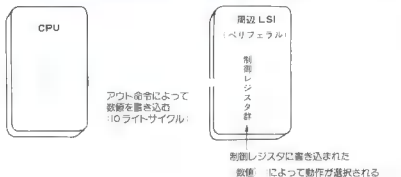
## ペリフェラルのプログラミング

ペリフェラルを使うときは、データの入出力を行なう前に、ペリフェラルの内部レジスタへ情報を書き込んで、目的にあった動作をするように設定しなければなりません。機能がよいほど選択の幅が広がりますので、たくさん情報を与えなければなりません。情報は次々に書き込みます。情報の中に何の情報かを示すビットが設けられている場合もありますが、書き込む順序によって、意味が違ってしまふことがあります。マニュアルに記載された順序を守れば安全です。

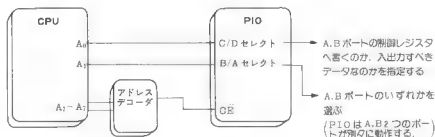
ペリフェラルのレジスタへ情報を書き込むには、そのペリフェラルのつながっている IO ポートへ出力命令を出します。PIO と SIO には、C/D セレクト信号端子があり、“1” のとき C すなわち制御語 (control word)、“0” のとき D すなわちデータと解釈しますので、制御情報を書き込むときはこの端子を“1”にして出力命令を出します。実際にはこの端子をアドレスバスのいずれかのビットへつなぎますと、制御語とデータとは別の IO ポートアドレスに配置されることとなります。CTC のチャンネルセレクトや PIO の B/A ポートセレクトも同じ考え方で対処します。

一度書き込んだ制御情報は、リセットされるか、または新たに書き込まれるまで有効です。また、特別な場合を除いて IN 命令で読み出すことはできません。





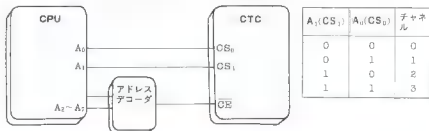
## PIO の場合



A <sub>7</sub> ~A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	番 地	ポート切換え	データ/制御語切換え
×	0	0	×0, ×4, ×8, ×C のどれか	B/A セレクト が"0"だから A ポート	C/D セレクトが"0"だからデータ
×	0	1	×1, ×5, ×9, ×D のどれか	B/A セレクト が"1"だから B ポート	// "1" // 制御語
×	1	0	×2, ×6, ×A, ×E のどれか	B/A セレクト が"0"だから A ポート	// "0" // データ
×	1	1	×3, ×7, ×B, ×F のどれか	B/A セレクト が"1"だから B ポート	// "1" // 制御語

×はアドレスデコードによってきまる

## CTC の場合

(CTC に対する入出力は制御レジスタにのみ対応する)  
(CS<sub>0</sub>, CS<sub>1</sub> は内部の 4 つのチャネルを選択する)

## PIO モード 0 の動作

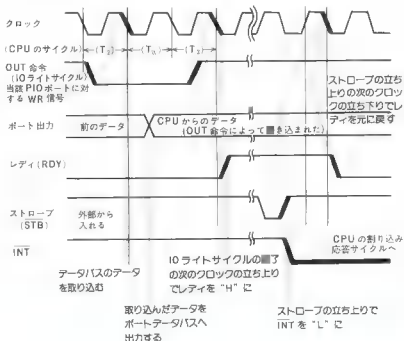
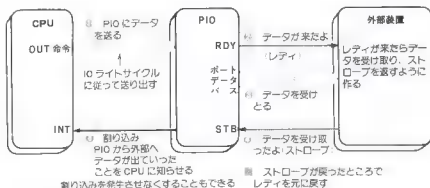
**モ**ード 0 は出力モードです。8 ビットの入出力線に出力[OUT] 命令で、出されたデータバスの状態がそのまま出てきます。データバスの内容が変わっても、次に出力命令がくるまで出力線の内容はそのままです。

CPU からの出力動作では、IO リクエスト ( $\overline{\text{IORQ}}$ ) とライト ( $\overline{\text{WR}}$ ) 信号が“L”になりますが、PIO にはライト信号の入力端子がありません。これは ( $\overline{\text{IORQ}}$ ) が“L”で、リード ( $\overline{\text{RD}}$ ) が“H”で、チップイネーブル ( $\overline{\text{CE}}$ ) が“L”で、 $\overline{\text{C/D}}$  がデータすなわち“L”のとき、PIO 内部でライト信号を作り出しているからです。

出力命令が出されると、データバスから PIO の出力レジスタへ信号が入ります。内部のライト信号が立ち上がると次のクロックの立ち下がりで、すでに出力線に出力レジスタの内容が乗り確定しています。同時にレディ信号線からレディ (RDY) 信号が出されます。外部装置はこのレディ信号が“H”になったことにより出力線のデータを取り込み、終わったらストローブ (STB) 信号線を“L”にして応答します。レディ信号は、このストローブの立ち上がりの次のクロックの立ち下がりで“L”に戻り、次のデータの出力命令を待ちます。割り込みがイネーブルにプログラムされていて、優先順位の高い割り込みがかかっていなければ、ストローブの立ち上がりで割り込み ( $\overline{\text{INT}}$ ) 出力は“L”になって CPU に割り込みをかけることができます。次の出力命令があるまで出力線の状態は変わりません。

レディ信号が“H”のとき、すなわち出力命令に対してストローブが返っていないときにさらに出力命令を出すと、レディ信号は一度“L”に戻ってすぐ“H”になります。レディ信号は正論理信号ですから“H”アクティブです。

レディとストローブの信号線をハンドシェーク線と呼びます。外部装置と CPU は PIO のハンドシェーク線を通じて、互いに確認し合いながら (同期をとりながら) データのやりとりを行なうわけです。

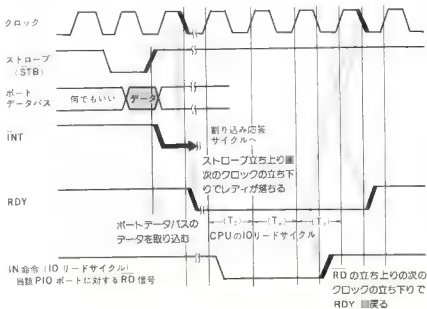
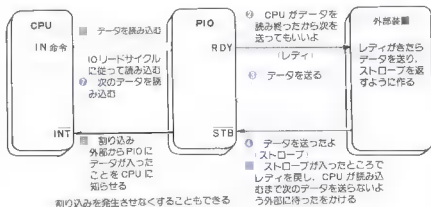


## PIO モード 1 の動作

**モ**ード 1 は入力モードです。外部から入ってきた 8 ビットの信号がポートの入出力線に与えられ、CPU からの入力 (IN) 命令でデータバスを通じて CPU の内部レジスタへ取り込みます。

レディ端子は、リセットがかかると“L”になっています。CPU からの入力命令で、“H”になって受け入れ可能を外部へ知らせます。この 1 発目の入力命令で読み込んだデータは無意味です。外部装置からレディ信号線が“H”のとき、入出力線にデータを与え、続いてストローブを“L”にするとストローブの立ち上がりで、PIO は入出力線から内部の入力レジスタへデータを取り込みます。次のクロックの立ち下がりで、レディを“L”にして外部から次のデータが入ることを禁止し、同時に割り込み可の状態であれば、CPU に割り込みをかけます。割り込み処理ルーチンで、入力 (IN) 命令を実行し、このデータを CPU の内部レジスタへ読み込みますと、リード ( $\overline{RD}$ ) 信号の立ち上がりで次のクロックの立ち下がりでレディ信号線を“H”に戻し、次のデータ入力に備えます。

モード 1 では 1 個のダミーの入力命令が必要です。プログラミングが終わり、外部からの信号を受け入れてもよい状態になったら [IN] 命令を実行して、レディ信号を出し外部装置に受け入れ可を知らせてください。レディ信号が受け入れ状態を示していないときに、入力線にデータを乗せてストローブ信号を与えても、入力レジスタ上へデータを書き込むことはできません。しかしなんらかの方法で CPU の入力命令の実行が遅れないようにしないと、データが失われるおそれがあります。

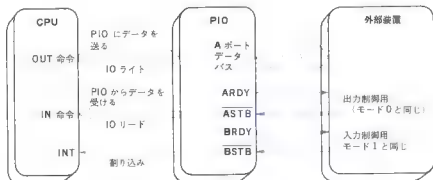


## PIO モード 2 の動作

**モ**ード 2 は A ポートだけについて可能な、入出力モードです。8 本のデータ線は入出力共用になります。ハンドシェーク線は A ポートが出力制御用、B ポートが入力制御用になります。したがって、B ポートはハンドシェーク線を必要としないモード 3 に設定しなければなりません。

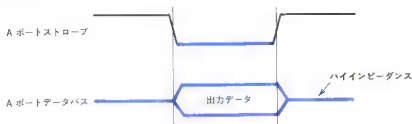
出力命令によるデータ出力動作はモード 0 と同じですが、モード 0 では出力線に常にデータが乗せられているのに対し、モード 2 のときは A ポートのストローブ ( $\overline{\text{ASTB}}$ ) 信号線が“L”になっている間だけ、多少遅れて乗せられてきます。外部装置はこのストローブを立ち上げるタイミングで、データを読み取ればよいのです。入力命令によるデータ入力動作は、B ポートのハンドシェーク線を使うほかはモード 1 と同じです。

割り込みをかけるタイミングも、モード 0、モード 1 と同じですが、入力動作では B ポートのハンドシェーク線を使うため、B ポートに書き込まれたベクトルが送出されます。したがって、入力、出力各動作で割り込みを別々のベクトルで制御することができますが、B ポートをビットモードで使用する場合は、本来の B ポートの動作で発生する割り込みに対するベクトルと、A ポートの入力動作で発生するベクトルが同じ値になってしまいます。いずれかの割り込み機能を使用しないようにするか、処理プログラムの中で、どちらの割り込みかを判断しなければなりません。



入力動作と出力動作ではベクトルが異なる一異なるように設定しておくこと

B ポートデータバスはモード3で使う、または使わない、ただしモード3に設定しておくこと



A ポートストロブが“L”のときだけ出力データがポートデータバスに現われる。したがってこのときは外部からデータを与えてはならない。他は、モード0、モード1と同じ。

## PIO モード 3 の動作

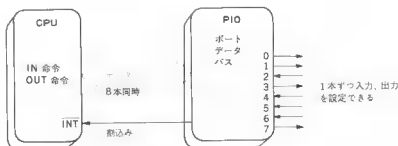
**モ**ード 3 はハンドシェイク線を使用しないで、非同期で入力出力を行います。ビット単位に入出力の設定ができ、割り込みは入力に指定したビットが指定した状態になったときに入ります。

出力命令で送られたデータは、モード 0 と同じタイミングで出力に指定されたビットに乗せられます。入力命令で読み込むと、リード ( $\overline{RD}$ ) 信号が立ち下がる直前の入力に指定されたビットの状態が CPU に読み込まれます。出力に指定されたビットは、それ以前に出力命令で送り出したままが読み込まれます。

入力に指定したビットは、さらに割り込みに関係させるか否かのマスク指定をすることができます。マスク指定で割り込み要因のモニタビットに指定したビットが全部そろったとき (AND 条件)、いずれか一つが入ったとき (OR 条件) を選択でき、入力線を正論理 (“H” アクティブ) としてとらえるか、負論理 (“L” アクティブ) としてとらえるかの選択もできます。

コントローラとしての用途では、PIO をモード 3 で使うことが多いと考えられます。周期的に CPU は入力ビットの状態を検査し、対応した出力信号を出力ビットへ乗せるようなプログラムを組みますが、入力ビットを読み込む周期はプログラムの長さで決まります。もっと早い対応が必要なシステムでは、割り込みを使うことにより解決できます。ただし、割り込みによるシステムの応答は、外部装置で起こり得る最悪の条件のときを想定して設計しないと、CPU が追いつかなくなることもあります。時間計算から許容ステップ数を出し、その範囲内のプログラムを作るのですが、システムクロック周波数の決定とも合わせて慎重に検討しなければなりません。





## 割り込み発生条件の設定

1. 入力ビットのうちどれを判断の基準とするか (マスク)
2. 入力は "H" アクティブか "L" アクティブか (論理)
3. AND か OR か (AND/OR)

たとえば、上の例で

- ・ビット 4 と 6 が共にアクティブになったとき割り込みを発生させたい
- ・入力は普通は "H" になって信号があるときだけ "L" になる ("L" アクティブ)

とすると

- ・マスク指定はビット 4 と 6
- ・論理は異論理
- ・AND/OR は AND

と指定する

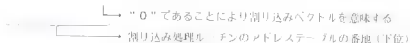
具体的には (次頁参照)

ベクトル	x x x x x x x 0	x x
モード設定	1 1 0 0 1 1 1 1	C F
ビット指定	0 1 1 1 0 1 0 0	7 4
	7ビット目      4ビット目	
割り込み制御語	1 1 0 1 0 1 1 1	D 7
マスク指定	1 0 1 0 1 1 1 1	A F

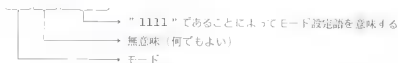
これを順次PIOの  
制御レジスタ群へ  
OUTする

## PIO のプログラミング

## 割り込みベクトル



## モード設定



7	6
---	---

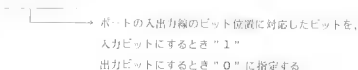
00 = モード0(出力モード)

01 = モード1(入力モード)

10 = モード2(双方向モード)

11 = モード3(ビットモード)

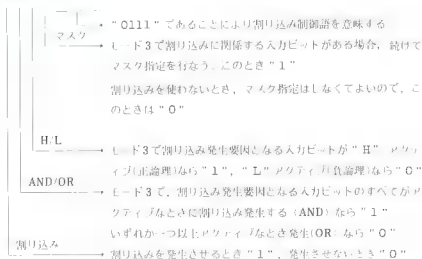
**ビット指定**      モード設定でモード3を指定したときは続けてこの指定をします。



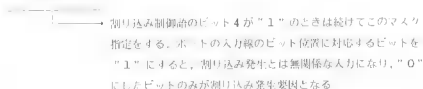
制御語の書き込みは A ポート、B ポートを別々に行なってください。  
A、B 混在させると、動作しない場合があります。順序もここに書かれていますと  
おりなら問題ありません。

割り込みを使わない場合は、ベクトルの書き込みは省略できます。

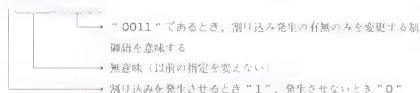
## 割り込み制御語



## マスク指定



## 割り込み制御語



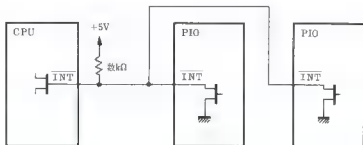
## PIO のプログラム例

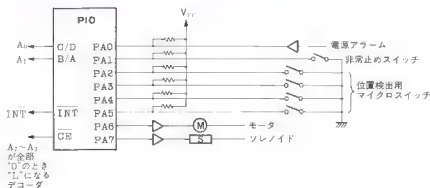
**P** IO をモード 3 で使用する場合の一例を考えてみます。PIO は IO ポートの 00H~03H に配置されているものとします。A ポートの 0~5 ビットを入力とし、他を出力ビットにします。電源異常信号と非常停止信号が共に“L”アクティブで、0 と 1 ビットにつながっています。処理に速度が要求されますので、割り込みをモード 2 で使用します。他の入力位置検出のマイクロスイッチの信号で、出力はすべてモータやソレノイドを駆動します。出力を“H”にすると動き、“H”にすると止まります。応答速度は問題にならないとします。

まず 0000H 番地からのルーチンで PIO を設定します。設定の最後で CPU の割り込み受け付けを可能にします。次に入力ビットを読み込み、その条件に従った制御情報を A レジスタにととのえ、出力ビットへ出力します。

割り込みが入ると、すべての出力を“L”にして動作を止めます。CPU はホルトに入り、新たにリセットがかかるのを待ちます。

PIO に限らず、ペリフェラルの  $\overline{\text{INT}}$  の出力端子はオープンドレインになっています。したがって、+5 V へ抵抗でプルアップしておかなければなりません (ワイアードオア)。





CONTROL		コメント行	PIO のプログラミング
ORG	0000H	: このプログラムをゼロ番地から配置	
LD	SP, 0000H	: スタックポインタイニシャライズ	
IM	2	: 割り込みモード	
LD	A, 01H	: レジスタセット	
LD	I, A		
LD	A, 00H	: ベクトル	
OUT	(01H), A		
LD	A, 0CFH	: PIO モード設定	
OUT	(01H), A		
LD	A, 3FH	: ビット指定	
OUT	(01H), A		
LD	A, 97H	: 割り込み制御語	
OUT	(01H), A		
LD	A, 0FCH	: マスク指定	
OUT	(01H), A		
EI		: 割り込み許可	
JOBS	1	: 仕事を始める	
IN	A, (00H)	: 入力ビットを読む	
OUT	(00H), A	: 出力をコントロールする	
JP	JOBS	: 次の仕事を始める	
ORG	0100H		
DEFW	INTR	: 割り込み処理ルーチンの先頭番地定義	
		: 割り込み処理ルーチン	
INTR	LD A, 0H	: モータを止める	
	OUT (00H), A		
	HALT	: CPU を止める。リセット信号が入るまで停止	
	END	: プログラム終り	

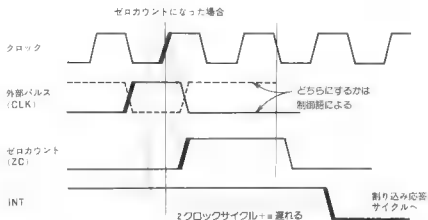
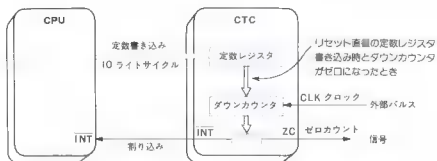
## CTC カウンタモード

**C** TCのクロック/トリガ(CLK/TRG)端子に、与えられるパルスをカウントします。あらかじめ設定された数を1個のパルスで1ずつ減らしてゼロになると、次のクロックの立ち上がりでゼロカウント/タイムアウト(ZC/TO)端子を1.5クロックの間“H”にし、割り込みを発生します。

カウントするパルスはシステムクロックの2倍以上の周波で、“H”と“L”の間は、おおよそ120ns以上必要です。パルスの立ち上がりまたは立ち下がりエッジ(指定できる)がくると、次のクロックの立ち上がりで、ダウンカウンタを1減じます。ダウンカウンタがゼロになるとゼロカウント信号を出し割り込みをかけます。同時に定数レジスタに設定されている数値をダウンカウンタへ書き込み、次のカウントに備えます。カウント途中で定数レジスタに新しい数値を書き込むと、次のゼロカウント時から有効になります。定数レジスタからダウンカウンタへの書き込みは、リセット状態から最初に定数レジスタが設定されたときと、ダウンカウンタがゼロになったときです。定数レジスタは8ビットですから1~256の値が設定できます。0を設定したときは256を意味します。

割り込みベクトルレジスタは、CTC 1個=4チャンネル分に1個しかありません。ベクトルの設定はチャンネル0に書き込みます。割り込みが発生すると要求するチャンネルによって、ビット1とビット2が決まったパターンに修飾されて送り出されてきます。優先順位はチャンネル0が一番高く、順にチャンネル3が最後位です。

制御語や定数の書き込み、読み出しのときのチャンネル指定は、チャンネルセレクト(CS<sub>0</sub>)と(CS<sub>3</sub>)を切り換えて行ないます。アドレスバスの2本(A<sub>0</sub>とA<sub>1</sub>がよく使われる)を接続すれば、CPU側からは四つのチャンネルが一連のポートアドレスに配列されます。PIOのC/Dセレクト、B/Aセレクトと同じ考え方です。



## CTC タイマモード

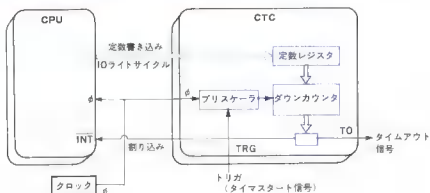
**カ**ウンタモードでは、不定期的なパルスをカウントしますが、決まった周期を持ったパルスをカウントすることにより、時間経過を知ることができます。これがタイマモードの動作の基本です。カウントするパルスはプリスケアラで、1/16 か 1/256 に分周されたシステムクロックを使います。システムクロックの周期と分周数と、定数レジスタに書かれた数値の積がゼロカウント/タイムアウト (ZC/TO) 端子へ信号が出てくる時間周期となります。

タイマの起動は、トリガと自動が選択できます。トリガ起動の場合は、クロック/トリガ端子の立ち上がりか立ち下がりのエッジ (選択できる) から2回目のクロックの立ち上がりでスタートします。自動の場合は、時間定数を書き込む出力命令の次の命令サイクルと同時にスタートします。

割り込みの発生と、定数の再設定についてはカウンタモードと同じです。

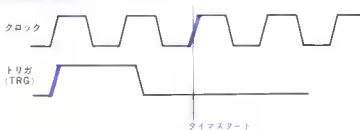
CTC はカウント中に、読み出しても差しかえありません。ダウンカウンタの現在値を知ることができます。ゼロカウント/タイムアウトを検知するのに ZC/TO 信号も割り込みも使わず、ダウンカウンタの値を読み出し、ゼロかどうか調べる方法が考えられますが、ダウンカウンタがゼロになり、次に定数レジスタの値が再びダウンカウンタへ書き込まれるまで、2クロックしかないので、この間に CPU が読み出すとは限らないため、不確実で実用できません。



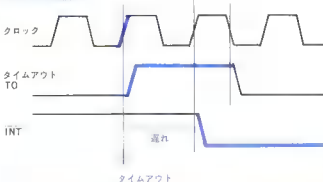


前項では  $\phi$  は省略してあるが、カウンタモードでもクロックは与えなければならない

#### トリガ起動の場合



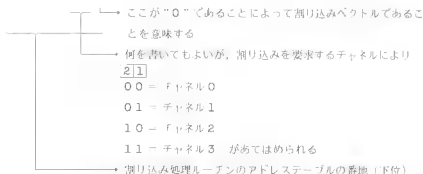
#### タイムアウトの場合



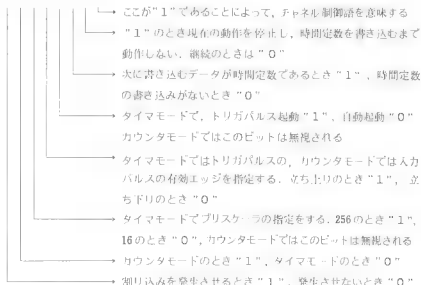
## CTC のプログラミング

### 割り込みベクトル

(チャンネル 0 に対してのみ書き込むことができます)



### チャンネル制御語 (各チャンネルごとに書き込みます)



## 時間定数

$$2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$$

→ チャンネル制御語のビット 7 が "1" のとき続けて書き込む。各ビットは 2 進数でゼロカウントになるまでの数値を表現している。ゼロのときは 256 として扱われる

## 8255 との接続

Z-80 ファミリには、パラレル入出力ポートとして PIO が用意されて、ハンドシェイクや割り込み機能など使いやすい機能が盛り込まれています。しかしメカニズムの制御で使われる入出力は、単純なビット単位のオンオフが多く、このような用途の場合はいろいろな機能よりむしろ、1 本でも多くのポートを持っていることが選択の条件になります。インテル社の 8255 は、PIO のような割り込みの機能は持っていませんが、8 ビットポートが 3 本と、PIO より多くのポートを持っており、上のような用途によく使われています。このような応用には、IC の数が増えるのをいとわなければ、標準ロジック IC の D-フリップフロップなどを使っても構成できます。

8255 は Z-80 ファミリとは設計メーカーも違い、1 世代前の製品ということもあって、完全な結合はできません。しかし機能を限定することによって、十分目的を達することが可能です。



## 例題

- プログラム 1 (ループ)
- プログラム 2 (判断)
- プログラム 3 (メモリクリアサブルーチン)
- プログラム 4 (変換テーブルサーチ)
- プログラム 5 (スイッチの表示)
- プログラム 6 (スイッチの表示-割り込み)

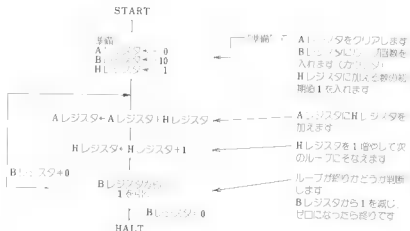
## プログラム 1 (ループ)

1 から 10 までの整数を加えるプログラムを考えます。答は A レジスタに入っていればよいとします。〈SAMPLE 1〉



しかし、A レジスタに何が入っているかわからない・RESET でも A レジスタはゼロにならない・また、A レジスタをゼロクリアします。A レジスタをゼロにするには、LD A, 0 でもよいが、2 バイト命令なので、A と A の排他的論理和をとることによりゼロにします。これなら 1 バイト命令です。

これが 1~100 までだったら上のやり方では大変です。そこで「くり返し」を使います。〈SAMPLE 2〉



この方法なら、B レジスタに入れる数を変えればいくつでも加えられます。ただし、このままでは A レジスタがオーバーフローすることがありますので工夫が必要です。

[注] サンプルプログラムを全部一連にしてアセンブルしたものです。各サンプルプログラム間に関連はありません。実行するときにはモニタプログラムを持ったワンボードコンピュータ、たとえばシャープの SM-B-80TE を使ってください。

行番号

1  
2  
3  
4

プログラムを  
見やすくする  
ためのラベル

\*\*\*\*\*

; SAMPLE 1

;

;

アトリブ マシン語 ラベル コード オペランド

5  
6 0000 AF  
7 0001 C601  
8 0003 C602  
9 0005 C603  
10 0007 C604  
11 0009 C605  
12 000B C606  
13 000D C607  
14 000F C608  
15 0011 C609  
16 0013 C60A  
17 0015 76

ORG 0000H  
XOR A  
ADD A, 1  
ADD A, 2  
ADD A, 3  
ADD A, 4  
ADD A, 5  
ADD A, 6  
ADD A, 7  
ADD A, 8  
ADD A, 9  
ADD A, 10  
HALT

← このプログラムを 0000H 番地から配置することをアセンブラに知らせるアセンブラ命令  
A をゼロにする  
A ← A + 1 A は 1 になる  
A ← A + 2 A は 3 になる  
A ← A + 3 A は 6 になる  
← ここで 1 ~ 10 までの数が A レジスタに入っている  
← CPU を止める

\*\*\*\*\*

; SAMPLE 2

;

;

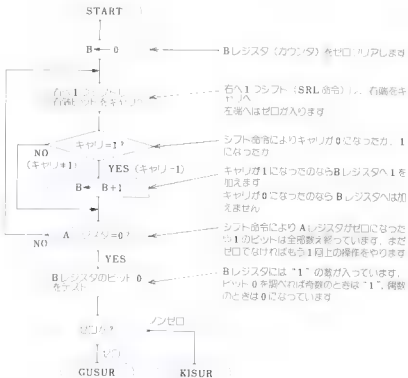
XOR A  
LD B, 10  
LD H, 1  
LOOP1 ADD A, H  
INC H  
DJNZ LOOP1  
HALT

カウンタの初期設定  
ポインタの初期設定  
A ← A + H  
H ← H + 1  
B ← B - 1 | B がゼロなら LOOP1 へ通へ、  
| ゼロなら次の処理へ

18  
19  
20  
21  
22 0016 AF  
23 0017 060A  
24 0019 2601  
25 001B 84  
26 001C 24  
27 001D 10FC  
28 001F 76

## プログラム 2 (判断)

A レジスタに入っているデータのビットパターンのうち“1”の数がいくつあるか数えます。結果が偶数なら GUSUR 番地へ、奇数なら KISUR 番地へジャンプするようにします。 (SAMPLE 3)



もっと簡単な方法があります。A と A の AND をとれば A は変わらずに、そのときの A の 1 の数に応じてパリティフラグがセットされます。

```

AND  A      ← A と A の AND により F レジスタをセットします
JP   PE, GUSUR ← PE は「パリティオープンすなわち A レジスタの「1」の数  
                が偶数であればジャンプせよ」の条件にジャンプ命令です
JP   KISUR
    
```

これでも上と同じ働きをします。F レジスタだけ変わりますが、他のレジスタの内容はこのルーチンに入る前そのまま変化しません。



```

29      ;*****
30      ;SAMPLE 3
31      ;
32      ;
33      0200      GUSUR      EQU      0200H ← EQUはGUSURというラベル名を0200H番地へ
34      025F      KISUR      EQU      025FH      ;結びつけるためのアセンブラ命令
35      0020 0600      LD      B,0
36      0022 CB3F      LOOP2   SRL      A
37      0024 3001      JR      NC,JP1
38      0026 04        INC      B
39      0027 20F9      JP1      JR      NZ,LOOP2
40      0029 CB40      BIT      0,B
41      002B CA0002     JP      Z,GUSUR
42      002E C35F02     JP      KISUR

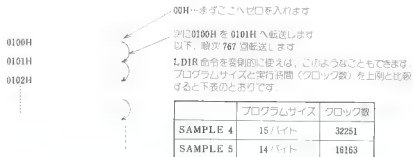
```

## プログラム 3 (メモリクリアサブルーチン)

0100H 番地から 03FFH 番地までの 300H バイト (=768 バイト) の RAM をゼロクリア (すべて 0 で埋めつくす) するサブルーチンを作ります。 (SAMPLE 4)



別の方法もあります。 (SAMPLE 5)



(プログラムサイズが小さく、実行が早いのがよいプログラムの条件です。ただし、あまり個人的なプログラムを作ると、あとで見たときに何をやっているのかわからなくなることがあります。メンテナンスやデバッグをしやすいのも、よいプログラムの条件です)

```

43      ;*****
44      ;SAMPLE 4
45      ;
46      ;
47 0031 210001 MCLRR LD HL,0100H
48 0034 010003 LD BC,0300H
49 0037 3600 ZEROM LD (HL),0
      0039 23 INC HL
51 003A 00 DEC BC
52 003B 70 LD A,B ; B, C 共にゼロであることを調べる
53 003C 01 OR C
54 003D 20FB JR NZ,ZEROM
55 003F C9 RET

```

```

56      ;*****
57      ;SAMPLE 5
58      ;
59      ;
60 0040 210001 MCLR1 LD HL,0100H
61 0043 110101 LD DE,0101H
62 0046 01FF02 LD BC,767 ← 10進数で定義している
63 0049 3600 LD (HL),0 ; 16進数では 02FF になっている
64 004B EDB0 LDIR
65 004D C9 RET

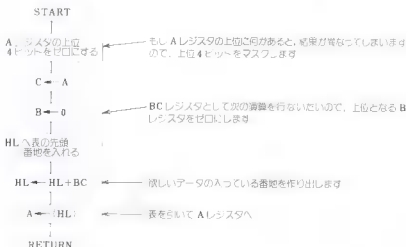
```

## プログラム 4 (変換-テーブルサーチ)

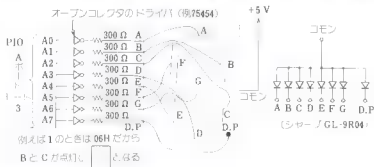
A レジスタの内容を表に従って変換します (下位 4 ビット対象)。 (SAMPLE 6)

A レジスタ	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
変換後	5C	06	5B	4F	66	6D	7D	27	7F	6F	77	7C	39	5E	79	71

変換前のデータが規則的では不規則です。したがって、前をメモリアドレスに対応させ、後をその内容とすれば、一発で表引きができます。もし相方共不規則なら一つ一つ致すかどうか調べて行くサーチの手法をとらなければなりません。



これは 4 ビットのデータを数字表示 LED に、16 進数表示するときを使うサブルーチンで、多桁表示したいときは、コモンを順次切り換えてダイナミック点灯させます。



```

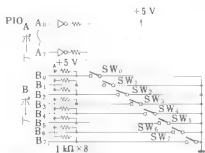
66                                     ;*****
67                                     ;SAMPLE6
68                                     ;
69                                     ;
70 004E E60F      CONV      AND      0FH  ← 上位4ビットをマスク
71 0050 4F        LD        C,A
72 0051 0600      LD        B,0
73 0053 215900    LD        HL, TABLE
74 0056 09        ADD       HL, BC  ← HLに答の入っているアドレスを作成
75 0057 7E        LD        A,(HL)
76 0058 C9        RET
77 0059 5C      TABLE DEF8      5CH
78 005A 06      DEF8      06H
79 005B 5B      DEF8      5BH
80 005C 4F      DEF8      4FH
81 005D 66      DEF8      66H
82 005E 6D      DEF8      6DH
83 005F 7D      DEF8      7DH
84 0060 27      DEF8      27H
85 0061 7F      DEF8      7FH
86 0062 6F      DEF8      6FH
87 0063 77      DEF8      77H
88 0064 7C      DEF8      7CH
89 0065 39      DEF8      39H
90 0066 5E      DEF8      5EH
91 0067 79      DEF8      79H
92 0068 71      DEF8      71H

```

テーブル(表):  
「DEFB」はオペランドに書かれた1バイトの値をそのままメモリ(読取り)でメモリに定義するためのアセンブラ命令。

## プログラム 5 (スイッチの表示)

PIO につながれた 8 個のスイッチのパターンを読み、前項のサブルーチンを使って LED に 16 進数表示させます。(SAMPLE 7)

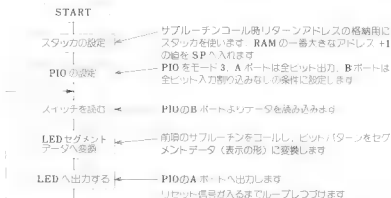


PIO のポートアドレスは下表とします

A ポート	データ	D0
A ポート	コントロール	D1
B ポート	データ	D2
B ポート	コントロール	D3



\*SW<sub>4</sub>~SW<sub>7</sub>は、この例では使わない(無視される)



IN 命令を実行するたびに、その時点の SW<sub>0</sub>~SW<sub>3</sub> の状態が 16 進数で LED に表示されます。



なら 0010 だから



と表示されます

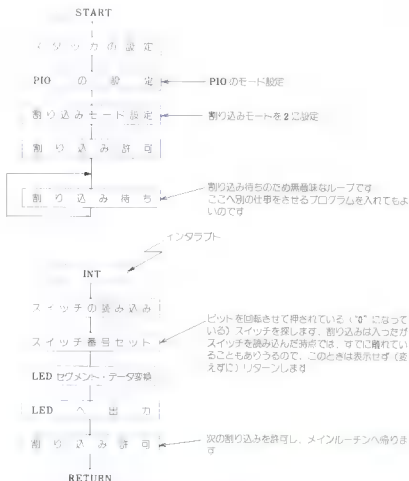
```

93          ;*****
94          ;SAMPLE7
95          ;
96          ;
97          0000      PIOAD1 EQU 0D0H      ;ポートアドレスのラベルの定義
98          0001      PIOAC1 EQU 0D1H      ;EQUは定義のためのアセンブラ命令
99          0002      PIOBD1 EQU 0D2H      ;レジスタには変換されない
100         0003      PIOBC1 EQU 0D3H
101         0069 310000      LD      SP,0000H ← スタックポインタ、FFFFH番地より前
102         006C 210000      LD      HL,INTPA
103         006F 0603      LD      B,3      ;PIOAポート、制御語を渡すため
104         0071 0E01      LD      C,PIOAC1
105         0073 ED03      OTIR
106         0075 210000      LD      HL,INTPB
107         0078 0603      LD      B,3      ;PIOBポート、制御語を渡すため
108         007A 0E03      LD      C,PIOBC1
109         007C ED03      OTIR
110         007E D0D2      LOOP7 IN      A,<PIOBD1> ← データを読み込み
111         0080 C04E00      CALL  CONU ← コンバータ変換モード
112         0083 D3D0      OUT     <PIOAD1>,A ← 出力
113         0085 C37E00      JP      LOOP7
114         0088 CF          INTPA  DEFB 0CFH ← モード0
115         0089 00          DEFB 00H ← 全ビット出力
116         008A 07          DEFB 07H ← 読み込みなし
117         008B CF          INTPB  DEFB 0CFH ← モード0
118         008C FF          DEFB 0FFH ← 全ビット出力
119         008D 07          DEFB 07H ← 読み込みなし

```

## プログラム 6 (スイッチの表示-割り込み)

前項のスイッチを押しボタンスイッチとして、押されたスイッチ番号 (SW<sub>0</sub> なら 0, SW<sub>3</sub> なら 3, SW<sub>7</sub> なら 7) を表示するように、割り込みを使ってプログラムを作りよ。ただし、同時に押したときは、先に押したほうを優先します。全く同時なら番号の小さいほうを優先します。離しても次に押されるまで表示し続けます。〈SAMPLE 8〉





```

120 ; *****
121 ; SAMPLE0
122 ;
123 ;
124 0000 PLOAD EQU 000H
125 0001 PIOAC EQU 001H
126 0002 PIOBD EQU 002H
127 0003 PIOBC EQU 003H
128 000E 310000 LD SP, 0000H
129 0091 21AC00 LD HL, INTLPA
130 0094 0603 LD B, 3
131 0096 0E01 LD C, PIOAC
132 0098 E0B3 OTIR
133 009A 21AF00 LD HL, INTLPB
134 009D 0605 LD B, 5
135 009F E0D3 LD C, PIOBC
136 00A1 E0B3 OTIR
137 00A3 3E01 LD A, 01H
138 00A5 ED47 LD I, A
139 00A7 ED5E IM 2
140 00A9 FB EI
141 00AA 10FE WAIT JR WAIT
142 00AC CF INTLPA DEFB 0CFH
143 00AD 00 DEFB 00H
144 00AE 07 DEFB 07H
145 00AF 00 INTLPB DEFB 00H
146 00B0 CF DEFB 0CFH
147 00B1 FF DEFB 0FFH
148 00B2 97 DEFB 97H
149 00B3 00 DEFB 00H
150 00B4 00 DRG 0100H
151 0100 0201 DEFW INT
152 0102 0B02 INT IN A, (PIOBD)
153 0104 0E00 LD C, 0
154 0106 0600 LD B, 0
155 0108 1F LOOPB RRA
156 0109 3006 JR NC, DISP
157 010B 0C INC C
158 010C 10FA DJNZ LOOPB
159 010E FB EI
160 010F ED40 RETI
161 0111 79 DISP LD A, C
162 0112 CD4E00 CALL CONU
163 0115 0300 OUT (PIOBD), A
164 0117 FB EI
165 0118 ED40 RETI
166 ; *****
167 END

```

スピードテストのプログラムを定義  
 スタックを設定、FFFFH まで 1 年前へ  
 PIO A ポート、初期値の書き込み  
 PIO B ポート、初期値の書き込み  
 I/O ステータス 01H  
 I/O ステータス 2  
 割り込み許可  
 割り込み待ちのループ  
 全ビット出力  
 割り込みなし  
 ステータス 00H  
 ステータス 3  
 全ビット入力  
 ステータス 1  
 ステータス 2  
 ステータス 3  
 [注] 下記参照  
 ステータス読み込み  
 右に 1 ビット右端の 1 をキープ  
 キーボードへの 70H を表示  
 I/O ステータス 00H、ステータス 1 のとき 1 ビット  
 リターン  
 データ変換をコール  
 (PIOBD) に A を 16 ビット出力  
 アセンブリの終了をアセンブラに知らせる  
 アセンブリ命令

- 1 割り込みあり、マスク制御あり、オフ、ローアディマ
  - 2 マスク制御なし、全ビット割り込み発生
  - 3 I/O ステータスベクトルにより取り扱われる番地
  - 4 割り込み処理ルーチンのエントリアドレス 0102 番地
- DEFW はオーバーランドの 2 バイトデータを定義するアセンブリ命令



# 付 録

## 付 1 Z-80 命 令 表

- **n** は 8 ビット定数
- **l m** は 16 ビット定数。**l** が上位 8 ビット、**m** が下位 8 ビット  
(ニーモニックではラベル名を書いてもよい)
- **d** は -128 ~ +127 までのディスプレイスメント
- **e** は -128 ~ +127 までのディスプレイスメント、次の命令の先頭番地を 0 とする。  
(ニーモニックではラベル名か絶対番地を書く、ただしアセンブラにより異なる場合もある)。
- **Cy** は キャリ (Cフラグ)
- 添字 **H** は上位 8 ビット、添字 **L** は下位 8 ビットを意味する。
- ビット操作命令 **SET RES BIT** で、ビット番号は下記のとおりである。



### ○ フラグ変化

- × 不定
- 1 1 になる
- 0 0 になる
- ・ 状態にしたがってセット、リセットされる
- 変化せず

ムーモニック	マシン語	フラグ変化						所要 クロック サイクル	動作
		S	Z	H	P/V	N	C		
ADC A, a	GE a							7	8ビット加算キヤリ付 (アド・ウィズ・キヤリ)
ADC A, A	8F								
ADC A, B	88								
ADC A, C	89								
ADC A, D	8A							4	A ← A + ソース + Cy
ADC A, E	8B								
ADC A, H	8C								
ADC A, L	8D								
ADC A, (HL)	8E								
ADC A, (IX+d)	DD 8E, d							7 19	
ADC A, (IY+d)	FD 8E, d							19	
ADC HL, BC	ED 4A								16ビット加算キヤリ付 (アド・ウィズ・キヤリ)
ADC HL, DE	ED 5A								
ADC HL, HL	ED 5A							15	HL ← HL + ソース + Cy
ADC HL, SP	ED 7A								
ADD A, a	C6 a							7	8ビット加算 (アド)
ADD A, A	87								
ADD A, B	80								
ADD A, C	81								
ADD A, D	82							4	A ← A + ソース
ADD A, E	83								
ADD A, H	84								
ADD A, L	85								
ADD A, (HL)	86								
ADD A, (IX+d)	DD 86, d							7 19	
ADD A, (IY+d)	FD 86, d							19	
ADD HL, BC	09								16ビット加算 (アド)
ADD HL, DE	19								
ADD HL, HL	29							11	HL ← HL + ソース
ADD HL, SP	39								
ADD IX, BC	DD 09								
ADD IX, DE	DD 19								
ADD IX, IX	29								IX ← IX + ソース
ADD IX, SP	DD 39								
ADD IY, BC	FD 09							15	
ADD IY, DE	FD 19								
ADD IY, IY	FD 29								IY ← IY + ソース
ADD IY, SP	FD 39								
AND a	86, a							7	論理積 (アンド)
AND A	A7								
AND B	A0								
AND C	A1								
AND D	A2							4	A ← A ∧ ソース
AND E	A3								
AND H	A4								
AND L	A5								
AND (HL)	A6							7	
AND (IX+d)	DD A6, d							19	
AND (IY+d)	FD A6, d							19	

a	b	作
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシン語	フラグ変化						所置 ラック ヤイタ	動作
		S	Z	H	P/V P/V	N	O		
BIT 0.A	CB 47								ビットテスト  ソースの第0ビットを調べ Zフラグを設定
BIT 0.B	CB 40								
BIT 0.C	CB 41								
BIT 0.D	CB 42								
BIT 0.E	CB 43								
BIT 0.H	CB 44								
BIT 0.L	CB 45								
BIT 0.(HL)	CB 46							12	
BIT 0.(IX+d)	DD CB,d,46							20	
BIT 0.(IY+d)	FD CB,d,46							20	
BIT 1.A	CB 4F								ビットテスト  ソースの第1ビットを調べ Zフラグを設定
BIT 1.B	CB 48								
BIT 1.C	CB 49								
BIT 1.D	CB 4A								
BIT 1.E	CB 4B								
BIT 1.H	CB 4C								
BIT 1.L	CB 4D								
BIT 1.(HL)	CB 4E							12	
BIT 1.(IX+d)	DD CB,d,4E							20	
BIT 1.(IY+d)	FD CB,d,4E							20	
BIT 2.A	CB 57								ビットテスト  ソースの第2ビットを調べ Zフラグを設定
BIT 2.B	CB 50								
BIT 2.C	CB 51								
BIT 2.D	CB 52								
BIT 2.E	CB 53								
BIT 2.H	CB 54								
BIT 2.L	CB 55								
BIT 2.(HL)	CB 56							12	
BIT 2.(IX+d)	DD CB,d,56							20	
BIT 2.(IY+d)	FD CB,d,56							20	
BIT 3.A	CB 5F								ビットテスト  ソースの第3ビットを調べ Zフラグを設定
BIT 3.B	CB 58								
BIT 3.C	CB 59								
BIT 3.D	CB 5A								
BIT 3.E	CB 5B								
BIT 3.H	CB 5C								
BIT 3.L	CB 5D								
BIT 3.(HL)	CB 5E							12	
BIT 3.(IX+d)	DD CB,d,5E							20	
BIT 3.(IY+d)	FD CB,d,5E							20	
BIT 4.A	CB 67								ビットテスト  ソースの第4ビットを調べ Zフラグを設定
BIT 4.B	CB 60								
BIT 4.C	CB 61								
BIT 4.D	CB 62								
BIT 4.E	CB 63								
BIT 4.H	CB 64								
BIT 4.L	CB 65								
BIT 4.(HL)	CB 66							12	
BIT 4.(IX+d)	DD CB,d,66							20	
BIT 4.(IY+d)	FD CB,d,66							20	

ムービー	マシン語	フラグ変化					アロックス サイタル	動作
		S	Z	H	P/V	N/C		
BIT 5.A	CB 8F							ビットテスト
BIT 5.B	CB 88							
BIT 5.C	CB 89							
BIT 5.D	CB 8A						8	ソースの第5ビットを調べ Zフラグを設定
BIT 5.E	CB 8B							
BIT 5.H	CB 8C	S	1			0		
BIT 5.L	CB 8D							
BIT 5.(HL)	CB 8E						12	
BIT 5.(IX+d)	DD CB d, 8E						20	
BIT 5.(IY+d)	FD CB d, 8E						20	
BIT 6.A	CB 77							ビットテスト
BIT 6.B	CB 70							
BIT 6.C	CB 71							
BIT 6.D	CB 72						8	ソースの第6ビットを調べ Zフラグを設定
BIT 6.E	CB 73							
BIT 6.H	CB 74	S	1			0		
BIT 6.L	CB 75							
BIT 6.(HL)	CB 76						12	
BIT 6.(IX+d)	DD CB d, 76						20	
BIT 6.(IY+d)	FD CB d, 76						20	
BIT 7.A	CB 7F							ビットテスト
BIT 7.B	CB 78							
BIT 7.C	CB 79							
BIT 7.D	CB 7A						8	ソースの第7ビットを調べ Zフラグを設定
BIT 7.E	CB 7B							
BIT 7.H	CB 7C	S	1			0		
BIT 7.L	CB 7D							
BIT 7.(HL)	CB 7E						12	
BIT 7.(IX+d)	DD CB d, 7E						20	
BIT 7.(IY+d)	FD CB d, 7E						20	
CALL NZ, i	C4 = i							成立時: サブリーチン・コール(条件付)
CALL Z, i	CC = i						17	条件が成立すれば宛番地 [PC] をスタックへPUSHし i へジャンプ [PC ← i]
CALL NC, i	D4 = i							不成立
CALL C, i	DC = i						10	成立しなければ本命令は無視する
CALL PQ, i	E4 = i							
CALL PE, i	EC = i							
CALL P, i	F4 = i							
CALL M, i	FC = i							
CALL i	CD = i						17	サブリーチン・コール(無条件) PC をスタックへPUSHし PC ← i
CCP	3F						4	Cy を反転 [Cy ← Cy]
CP =	FE =						7	比較 (コンペア)
CP A	BF							A ソースの演算をする
CP B	B8							A の内容は変わらないがフラグだけが変化
CP C	B9							
CP D	BA						4	
CP E	BB					1		
CP H	BC							
CP L	BD							
CP (HL)							7	
CP (IX+d)	DD BE d						19	
CP (IY+d)	FD BE d						19	

ユーモニツ	マシン語	フラグ変化					所要 クロック サイクル	動作
		S	Z	H	P/V	N/O		
CPD	BD A9	X	X	-	1	-	16	比較(コンペア・インクリメント) A ← HL のフラグ変化のみ HL ← HL-1 BC ← BC-1
CPDR	BD B9	X	X	-	1	-	16 1バイトにつき 21 最終の4 16	比較(コンペア・インクリメント・リポート) CPD を A ← HL [Zフラグ=1] または BC ← 0 [Vフラグ=0] までくり返す
CPI	BD A1	X	X	-	1	-	16	比較(コンペア・インクリメント) A ← HL のフラグ変化のみ HL ← HL-1 BC ← BC-1
CPIR	BD B1	X	X	-	1	-	16 1バイトにつき 21 最終の4 16	比較(コンペア・インクリメント・リポート) CPI を A ← HL [Zフラグ=1] または BC ← 0 [Vフラグ=0] までくり返す
CPL	2F	-	-	1	-	-	4	32ビット A レジスタに対してビット反転 0 → 1 1 → 0
DAA	27	-	-	-	-	-	4	テンソル(アキュムレータ)・アキュムレータ A レジスタに対して10進補正
DEC A	3D	-	-	-	-	-	4	8ビットデクリメント
DEC B	05	-	-	-	-	-	4	7ビットデクリメント
DEC C	0D	-	-	-	-	-	4	6ビットデクリメント
DEC D	15	-	-	-	-	-	4	5ビットデクリメント
DEC E	1D	-	-	-	-	-	4	4ビットデクリメント
DEC H	25	-	-	-	-	-	4	2ビットデクリメント
DEC L	2D	-	-	-	-	-	4	1ビットデクリメント
DEC (HL)	35	-	-	-	-	-	11	16ビットデクリメント
DEC (IX+d)	DD 35 d	-	-	-	-	-	23	16ビットデクリメント
DEC (IY+d)	FD 35 d	-	-	-	-	-	23	16ビットデクリメント
DEC BC	0B	-	-	-	-	-	5	16ビットデクリメント
DEC DE	1B	-	-	-	-	-	5	16ビットデクリメント
DEC HL	2B	-	-	-	-	-	5	16ビットデクリメント
DEC SP	3B	-	-	-	-	-	5	16ビットデクリメント
DEC IX	DD 2B	-	-	-	-	-	10	16ビットデクリメント
DEC IY	FD 2B	-	-	-	-	-	10	16ビットデクリメント
DI	F3	-	-	-	-	-	4	割り込み禁止(ディセンプル・インタラプト)
DJNZ e	1C e	-	-	-	-	-	8 5+0 13	割り込み禁止(ディセンプル・インタラプト) (デクリメント・ジャンプ・ジャンプ) B ← B-1 Bが0ならe(16ビット)にジャンプ [PC ← PC + e] B ← 0ならジャンプせず[e ← 0と同じ]
RI	FB	-	-	-	-	-	4	割り込み許可(イネーブル・インタラプト)
EX SP, HL	E3	-	-	-	-	-	19	交換(エクスチェンジ)
EX SP, IX	DD E3	-	-	-	-	-	23	7ビットデクリメント
EX SP, IY	FD E3	-	-	-	-	-	23	7ビットデクリメント
EX AF, AF	08	-	-	-	-	-	4	交換(エクスチェンジ)
EX DE, HL	EB	-	-	-	-	-	4	7ビットデクリメント
EXX	D9	-	-	-	-	-	4	BC DE HLの内容と BC DE HLの内容を交換する

ニーモニック	マシン語	フラグ変化					所要 クロック サイクル	動作
		S	Z	H	F/V P/V	N		
HALT	7B						4	命令実行の進行を止める。または割り込み待ちとなる(ホルト)。
IM 0	ED 46						8	割り込みモードを0に設定する。
IM 1	ED 56						8	割り込みモードを1に設定する。
IM 2	ED 5E						8	割り込みモードを2に設定する。 Bヒートインタリメント。
INC A	3C							
INC B	04							
INC C	0C							
INC D	14							
INC E	1C							
INC H	24					0		
INC L	2C							
INC (HL)	34						11	
INC (IX+d)	DD 34 d						23	
INC (IY+d)	FD 34 d						23	
INC BC	03							16ビットインタリメント
INC DE	13							
INC HL	23						6	グループモード1
INC SP	33							
INC IX	DD 23						10	
INC IY	FD 23						10	
IN A, (C)	ED 7B							入力 Cレジスタの内容をポータルからAレジスタへ
IN B, (C)	ED 40							Cレジスタの内容をポータルからBレジスタへ
IN C, (C)	ED 48							Cレジスタの内容をポータルからCレジスタへ
IN D, (C)	ED 50							Cレジスタの内容をポータルからDレジスタへ
IN E, (C)	ED 58							Cレジスタの内容をポータルからEレジスタへ
IN H, (C)	ED 60							Cレジスタの内容をポータルからHレジスタへ
IN L, (C)	ED 68							Cレジスタの内容をポータルからLレジスタへ
IN A, (B)	DB 0						11	入力 B番地のポートからAレジスタへ
IND	ED AA	X	X	X	X	X	16	インタリメント (HL) ← (C) HL ← HL + 1 B ← B + 1
INIR	ED BA	X	1	X	X	X	21 16	インタリメント・リセット INDをB+0までくり返す
INI	ED A2	X	X	X	X	X	16	インタリメント (HL) ← (C) HL ← HL + 1 B ← B + 1
INIR	ED B2	X	1	X	X	X	21 16	インタリメント・リセット INIをB+0までくり返す
JP (HL)	E9						4	ジャンプ(無条件)
JP (IX)	DD E9						8	Cレジスタの内容をポータルからジャンプ
JP (IY)	FD E9						8	(PC ← HL)
JP (B)	CB 0						10	B番地へジャンプ(PC ← B)



ニューモニック	マシン語	フラグ変化					所 要 ラ ッ ク サイ クル	備 考		
		S	Z	H	P/V P	N			C	
JP NZ, i	GZ, i							10	ジャンプ(条件付) ・条件が成立すれば10番地へジャンプ (PC+10) ・不成立なら本命令は無視する	
JP Z, i	CA, i									
JP NC, i	DZ, i									
JP C, i	DA, i									
JP PO, i	EZ, i									
JP PE, i	EA, i									
JP P, i	FZ, i							12	ジャンプ・リラティブ(条件付) ・eバイト先へジャンプする [PC+PC+e]	
JP M, i	FA, i									
JR	1B, e									
JR NZ, e	2D, e								条件成立 12 条件不成立 7	ジャンプ・リラティブ(条件付) ・条件が成立すればeバイト先へジャンプ [PC+PC+e] ・不成立なら本命令は無視する
JR Z, e	2E, e									
JR NC, e	3C, e									
JR C, e	3E, e									
LD A, a	3H, a							7	8ビット転送(ワード)	
LD A, A	7F							4	A ← R	
LD A, B	7B									
LD A, C	79									
LD A, D	7A									
LD A, E	7B									
LD A, H	7C									
LD A, L	7D							13 7 7 7	8ビット転送(ワード) A ← I A ← R IFF: 0のとき割り込み禁止(DI) 1のとき割り込み可(EI)にな っている LD A, I LD A, R ではこの値がP/Vにコピーされる	
LD A, (i)	3A, i									
LD A, (BC)	0A									
LD A, (DE)	1A									
LD A, (HL)	7E									
LD A, (IX+d)	DD 7E d									
LD A, (IY+d)	FD 7E d							19		
LD A, I	ED 57							9		
LD A, R	ED 5F							9		
LD B, a	0B, a							7	8ビット転送(ワード)	
LD B, A	47							4	B ← R	
LD B, B	40									
LD B, C	41									
LD B, D	42									
LD B, E	43									
LD B, H	44									
LD B, L	45							7 19 19		
LD B, (HL)	46									
LD B, (IX+d)	DD 46 d									
LD B, (IY+d)	FD 46 d									

ユニコード	マシン語	フラグ変化	所 要 クロック サイクル	動 作
		S Z H P/V N C		
LD C,n	OE n		7	8ビット転送(ロード)
LD C,A	4F			
LD C,B	48			
LD C,C	49			C←ソース
LD C,D	4A		4	
LD C,E	4B			
LD C,H	4C			
LD C,L	4D			
LD C,(HL)	4E		7	
LD C,(IX+d)	DD 4E d		19	
LD C,(IY+d)	FD 4E d		19	
LD D,n	16 n		7	8ビット転送(ロード)
LD D,A	57			
LD D,B	58			
LD D,C	51			D←ソース
LD D,D	52		4	
LD D,E	53			
LD D,H	54			
LD D,L	56			
LD D,(HL)	56		7	
LD D,(IX+d)	DD 56 d		19	
LD D,(IY+d)	FD 56 d		19	
LD E,n	1E n		7	8ビット転送(ロード)
LD E,A	5F			
LD E,B	58			
LD E,C	59			E←ソース
LD E,D	5A		4	
LD E,E	5B			
LD E,H	5C			
LD E,L	5D			
LD E,(HL)	5E		7	
LD E,(IX+d)	DD 5E d		19	
LD E,(IY+d)	FD 5E d		19	
LD H,n	26 n		7	8ビット転送(ロード)
LD H,A	67			
LD H,B	68			
LD H,C	61			H←ソース
LD H,D	62		4	
LD H,E	63			
LD H,H	64			
LD H,L	65			
LD H,(HL)	66		7	
LD H,(IX+d)	DD 66 d		19	
LD H,(IY+d)	FD 66 d		19	

ニーモニック	マシン語	フラグ変化	所 要 クロック サイクル	動 作
		S Z H P/V N C		
LD L, m	2E, n		7	8ビット転送 (ロード)
LD L, A	6F			
LD L, B	68			
LD L, C	69			
LD L, D	6A		4	L ← (A)
LD L, E	6B			
LD L, H	6C			
LD L, L	6D			
LD L, (HL)	6E		7	
LD L, IX-d	DD 6E d		19	
LD L, IY-d	FD 6E d		19	
LD I, A	ED 47		9	8ビット転送 I ← A
LD R, A	ED 4F		9	R ← A
LD (I), A	32 m f		13	8ビット転送 (I) ← A
LD (BC), A	02		7	(BC) ← A
LD (DE), A	12		7	(DE) ← A
LD (HL), a	36 n		10	8ビット転送 (ロード)
LD (HL), A	77			
LD (HL), B	70			
LD (HL), C	71			(HL) ← ソース
LD (HL), D	72		7	
LD (HL), E	73			
LD (HL), H	74			
LD (HL), L	75			
LD (IX-d), n	DD 36 d n			8ビット転送 (ロード)
LD (IX-d), A	DD 77 d			
LD (IX-d), B	DD 70 d			
LD (IX-d), C	DD 71 d			(IX-d) ← ソース
LD (IX-d), D	DD 72 d		19	
LD (IX-d), E	DD 73 d			
LD (IX-d), H	DD 74 d			
LD (IX-d), L	DD 75 d			
LD (IY-d), n	FD 36 d n			8ビット転送 (ロード)
LD (IY-d), A	FD 77 d			
LD (IY-d), B	FD 70 d			
LD (IY-d), C	FD 71 d			(IY-d) ← ソース
LD (IY-d), D	FD 72 d		19	
LD (IY-d), E	FD 73 d			
LD (IY-d), H	FD 74 d			
LD (IY-d), L	FD 75 d			
LD BC, (m)	01 m f		10	16ビット転送 (16ビットデータ)
LD BC, (m)	ED 4B m f		20	BC ← ソース
LD DE, (m)	11 m f		10	16ビット転送 (16ビットデータ)
LD DE, (m)	ED 5B m f		20	DE ← ソース
LD HL, (m)	21 m f		10	16ビット転送 (16ビットデータ)
LD HL, (m)	2A m f		16	HL ← ソース

ニーモニック		マシン語	フラグ変化					所 要 クロック サイクル	動 作
			S	Z	H	P/V	N	C	
LD	SP, $\text{f}\#$	31 0 $\text{f}$							10 16ビット転送
LD	SP, $\langle \text{f}\# \rangle$	ED 7B 0 $\text{f}$							20
LD	SP, HL	F9							6 SP ← PC
LD	SP, IX	DD FB							10
LD	SP, IY	FD FB							10
LD	IX, $\text{f}\#$	DD 21 0 $\text{f}$							14 16ビット転送
LD	IX, $\langle \text{f}\# \rangle$	DD 2A 0 $\text{f}$							26 IX ← ソース
LD	IY, $\text{f}\#$	FD 21 0 $\text{f}$							14 16ビット転送
LD	IY, $\langle \text{f}\# \rangle$	FD 2A 0 $\text{f}$							20 IY ← ソース
LD	$\langle \text{f}\# \rangle$ , BC	ED 43 0 $\text{f}$							20 16ビット転送 (ロード)
LD	$\langle \text{f}\# \rangle$ , DE	ED 53 0 $\text{f}$							20
LD	$\langle \text{f}\# \rangle$ , HL	22 0 $\text{f}$							16 $\langle \text{f}\# \rangle$ ← ソース L
LD	$\langle \text{f}\# \rangle$ , SP	ED 73 0 $\text{f}$							20 $\langle \text{f}\# + 1 \rangle$ ← ソース H
LD	$\langle \text{f}\# \rangle$ , IX	DD 22 0 $\text{f}$							20
LD	$\langle \text{f}\# \rangle$ , IY	FD 22 0 $\text{f}$							20
LDD		ED A8							ブロック転送 (ロード・ディクリメント) (DE) ← (HL) DE ← DE - 1 HL ← HL + 1 BC ← BC - 1
LDDR		ED B8							1-16 17-1 21 21 16 ブロック転送 (ロード・ディクリメント・リヒート) LDDをBC ← 0までくり返す
LDI		ED A0							ブロック転送 (ロード・インクリメント) (DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1
LDIR		ED B0							1-1 21 21 16 ブロック転送 (ロード・インクリメント・リヒート) LDI ← BC ← 0までくり返す
NEG		ED 44						1	ニーマイト Zの補数をとる A ← 0 - A
NOP		00							4 何もしないで次へ (ブランチ・ジャンプ)
OR	n	F6 0							7 論理和 (オア)
OR	A	B7							
OR	B	B0							
OR	C	B1							
OR	D	B2							
OR	E	B3							
OR	H	B4							
OR	L	B5							
OR	(HL)	B6							
OR	(IX+d)	DD B6 d							7
OR	(IY+d)	FD B6 d							19

a	b	答
0	0	0
0	1	1
1	0	1
1	1	1

ニーモニック	マシン語	ラダ変化					所要 クロック サイクル	動作							
		S	Z	H	P/V P/V	N									
OUT (C), A	ED 79							出力							
OUT (C), B	ED 41														
OUT (C), C	ED 49							各レジスタの内容をCレジスタの内容番地の							
OUT (C), D	ED 51						12	ポートへ出力							
OUT (C), E	ED 59							[(C)←PORS]							
OUT (C), H	ED 61														
OUT (C), L	ED 69														
OUT (B), A	D3 B						11	出力 Aレジスタの内容をB番地のポートへ							
OUTD	ED AB														
							16	アウト・インタリメント (C)←HL, HL←HL-1 B←B-1							
OTDR	ED BB						16 17 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000								アウト・インタリメント・リロード OUTDをB←Oでくり返す
OUTI	ED A3						16	アウト・インタリメント (C)←(HL), HL←HL-1 B←B-1							
OTIR	ED B3						16 17 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000								アウト・インタリメント・リロード OUTIをB←O1でくり返す
POP AF	F1							15ビット転送(ポップ)							
POP BC	C1							スタックからレジスタへ転送							
POP DE	D1							デスティネーション・SP							
POP HL	E1							デスティネーション・SP+1							
POP IX	DD E1							SP←SP+2							
POP IY	FD E1														
PUSH AF	F5							16ビット転送(プッシュ)							
PUSH BC	C5							レジスタからスタックへ転送							
PUSH DE	D5							SP←SP+2							
PUSH HL	E5							SP←SP+2							
PUSH IX	DD E5							SP←SP+2							
PUSH IY	FD E5														
RES 0, A	CB 87							ビットリセット							
RES 0, B	CB 80														
RES 0, C	CB 81														
RES 0, D	CB 82														
RES 0, E	CB 83														
RES 0, H	CB 84														
RES 0, L	CB 85														
RES 0, (HL)	CB 86														
RES 0, (IX+d)	DD CB, d, 86														
RES 0, (IY+d)	FD CB, d, 86														

ニ	モ ニ ッ タ	ア シ ャ 語	フ ラ タ 変 化					所 要 クロック サイクル	動 作
			S	Z	H	P/V	N	C	
RES	1. A	CB 8F							ビットリセット  ワースの第1ビット→0
RES	1. B	CB 8B							
RES	1. C	CB 89							
RES	1. D	CB 8A						8	
RES	1. E	CB 8B							
RES	1. H	CB 8C							
RES	1. L	CB 8D							
RES	1. HL	CB 8E						15	
RES	1. IX+d	DD CB d 8E						23	ビットリセット  ワースの第2ビット→0
RES	1. IY+d	FD CB d 8E						23	
RES	2. A	CB 97							
RES	2. B	CB 9D							
RES	2. C	CB 91							
RES	2. D	CB 92						8	
RES	2. E	CB 93							
RES	2. H	CB 94							
RES	2. L	CB 95							ビットリセット  ワースの第3ビット→0
RES	2. HL	CB 96						15	
RES	2. IX+d	DD CB d 96						23	
RES	2. IY+d	FD CB d 96						23	
RES	3. A	CB 9F							
RES	3. B	CB 9B							
RES	3. C	CB 99							
RES	3. D	CB 9A						8	
RES	3. E	CB 9B							ビットリセット  ワースの第4ビット→0
RES	3. H	CB 9C							
RES	3. L	CB 9D							
RES	3. HL	CB 9E						15	
RES	3. IX+d	DD CB d 9E						23	
RES	3. IY+d	FD CB d 9E						23	
RES	4. A	CB A7							
RES	4. B	CB A0							ビットリセット  ワースの第5ビット→0
RES	4. C	CB A1							
RES	4. D	CB A2						8	
RES	4. E	CB A3							
RES	4. H	CB A4							
RES	4. L	CB A5							
RES	4. HL	CB A6						15	
RES	4. IX+d	DD CB d A6						23	
RES	4. IY+d	FD CB d A6						23	ビットリセット  ワースの第6ビット→0
RES	5. A	CB AF							
RES	5. B	CB A8							
RES	5. C	CB A9							
RES	5. D	CB AA						8	
RES	5. E	CB AB							
RES	5. H	CB AC							
RES	5. L	CB AD							
RES	5. HL	CB AE						15	ビットリセット  ワースの第7ビット→0
RES	5. IX+d	DD CB d AE						23	
RES	5. IY+d	FD CB d AE						23	

ユーモニタ	マシン語	フラグ変化						動作
		S	Z	H	P/V P/V	N	0	
RES 6, A	CB B7							ビットリセット
RES 6, B	CB B0							
RES 6, C	CB B1							ビットの第6ビット→0
RES 6, D	CB B2							8
RES 6, E	CB B3							
RES 6, H	CB B4							
RES 6, L	CB B5							
RES 6, (HL)	CB B6							15
RES 6, (IX-d)	DD CB, d, B6							23
RES 6, (IY-d)	FD CB, d, B6							23
RES 7, A	CB B7							ビットリセット
RES 7, B	CB B0							
RES 7, C	CB B1							ビットの第7ビット→0
RES 7, D	CB B2							8
RES 7, E	CB B3							
RES 7, H	CB B4							
RES 7, L	CB B5							
RES 7, (HL)	CB B6							15
RES 7, (IX-d)	DD CB, d, B6							23
RES 7, (IY-d)	FD CB, d, B6							23
RET	C9							10 サブルーチンからのリターン PCスタックリPOP
RET NZ	CD							条件付リターン
RET Z	CA							・条件が成立すれば
RET NC	D0							条件成立 POP PC
RET C	D8							11 ・条件成立 POP PC
RET PO	E0							5 ・条件成立 POP PC
RET PE	E8							
RET P	F0							
RET M	F8							
RTI	ED 4D							14 割り込み処理からのリターン POP PC
RTN	ED 45							14 ノンマスク割り込み処理からのリターン POP PC
RLA	17			0			0	4 ローテート・シフト・アキュムレータ RL Aと同じ
RLCA	07			0			0	4 ローテート・シフト・アキュムレータ・C RLC Aと同じ
RRA	1F			0			0	4 ローテート・シフト・アキュムレータ RR Aと同じ
RRAA	0F			0			0	4 ローテート・シフト・アキュムレータ・C RRC Aと同じ

二 三 五 七 九	マシン語	フラグ変化					所 要 クロック サイクル	作
		S	Z	H	P/V P/V	N		
RL A	CB 17							ローカール・レジスタ
RL B	CB 10							
RL C	CB 11							
RL D	CB 12						8	
RL E	CB 13							
RL H	CB 14		0		0			Cy ← ? 0 ←
RL L	CB 15							
RL (HL)	CB 16						15	
RL (IX+d)	DD CB d, 16						23	
RL (IY+d)	FD CB d, 16						23	
RLC A	CB 07							ローカール・レジスタ・インクリメント
RLC B	CB 00							
RLC C	CB 01							
RLC D	CB 02						8	
RLC E	CB 03		0		0			Cy ← ? 0 ←
RLC H	CB 04							
RLC L	CB 05							
RLC (HL)	CB 06						15	
RLC (IX+d)	DD CB d, 06						23	
RLC (IY+d)	FD CB d, 06						23	
RR A	CB 1F							ローカール・レジスタ
RR B	CB 18							
RR C	CB 19							
RR D	CB 1A						8	
RR E	CB 1B		0		0			Cy → ? 0 →
RR H	CB 1C							
RR L	CB 1D							
RR (HL)	CB 1E						15	
RR (IX+d)	DD CB d, 1E						23	
RR (IY+d)	FD CB d, 1E						23	
RRC A	CB 0F							ローカール・レジスタ・デクリメント
RRC B	CB 08							
RRC C	CB 09							
RRC D	CB 0A						8	
RRC E	CB 0B		0		0			Cy → ? 0 →
RRC H	CB 0C							
RRC L	CB 0D							
RRC (HL)	CB 0E						15	
RRC (IX+d)	DD CB d, 0E						23	
RRC (IY+d)	FD CB d, 0E						23	
RLD	ED 6F		0		0		18	ローカール・レジスタ・ディシフト A 7 4 3 0 7 4 3 0 (HL) ↑ ↓
RRD	ED 67		0		0		18	ローカール・レジスタ・ディシフト A 7 4 3 0 7 4 3 0 (HL) ↑ ↓



操 作	ア ー ジ ャ	レジスタ	ア ラ タ 変 化					所 要 クロック サイクル	動 作
			■	Z	H	P/V	N/C		
RST 00H		C7							11 0000H・003EHのいずれかの番地 に対するCALL
RST 06H		CF							
RST 1CH		D7							
RST 1EH		DF							
RST 2CH		E7							
RST 2EH		EF							
RST 3CH		F7							
RST 3EH		FF							
SBC A,n		DE n						2	8ビット引き算(キャリ付) (サブラクト・ウィッチ・キャリ)
SBC A,A		9F							
SBC A,B		98						4	A←A-ソース-Cy
SBC A,C		99							
SBC A,D		9A						1	
SBC A,E		9B							
SBC A,H		9C						7	
SBC A,L		9D							
SBC A,(HL)		9E						19	
SBC A,(IX+d)		DD 9E d						19	
SBC A,(IY-d)		FD 9E d							15 16ビット引き算(キャリ付) (サブラクト・ウィッチ・キャリ) HL←HL-ソース-Cy
SBC HL,BC		ED 42							
SBC HL,DE		ED 52							
SBC HL,HL		ED 62							
SBC HL,SP		ED 72							
SCF		37		0			0 1	4	セット・キャリアラフ Cy←1
SET 0,A		CB C7						8	ビットセット ソースの第8ビット←1
SET 0,B		CB C0							
SET 0,C		CB C1						15	
SET 0,D		CB C2							
SET 0,E		CB C3						23	
SET 0,H		CB C4							
SET 0,L		CB C5						23	
SET 0,(HL)		CB C6							
SET 0,(IX+d)		DD CB d C6							ビットセット ソースの第1ビット←1
SET 0,(IY-d)		FD CB d C6							
SET 1,A		CB CF						15	
SET 1,B		CB C8							
SET 1,C		CB C9						23	
SET 1,D		CB CA							
SET 1,E		CB CB						23	
SET 1,H		CB CC							
SET 1,L		CB CD						23	
SET 1,(HL)		CB CE							
SET 1,(IX+d)		DD CB d CE							
SET 1,(IY-d)		FD CB d CE							

ユーモニツタ	マシン語	フラグ変化					所 要 クロック サイクル	動 作
		S	Z	H	P/V P/V	N		
SET 2.A	CB D7							ビットセット
SET 2.B	CB D0							
SET 2.C	CB D1							8
SET 2.D	CB D2							
SET 2.E	CB D3							15
SET 2.H	CB D4							
SET 2.L	CB D5							23
SET 2.HL	CB D6							
SET 2. IX-d	DD CB d D6							23
SET 2. IY-d	FD CB d D6							
SET 3.A	CB D7							ビットセット
SET 3.B	CB D8							
SET 3.C	CB D9							8
SET 3.D	CB DA							
SET 3.E	CB DB							15
SET 3.H	CB DC							
SET 3.L	CB DD							23
SET 3.HL	CB DE							
SET 3. IX-d	DD CB d DE							23
SET 3. IY-d	FD CB d DE							
SET 4.A	CB E7							ビットセット
SET 4.B	CB E0							
SET 4.C	CB E1							8
SET 4.D	CB E2							
SET 4.E	CB E3							15
SET 4.H	CB E4							
SET 4.L	CB E5							23
SET 4.HL	CB E6							
SET 4. IX-d	DD CB d E6							23
SET 4. IY-d	FD CB d E6							
SET 5.A	CB EF							ビットセット
SET 5.B	CB E8							
SET 5.C	CB E9							8
SET 5.D	CB EA							
SET 5.E	CB EB							15
SET 5.H	CB EC							
SET 5.L	CB ED							23
SET 5.HL	CB EE							
SET 5. IX-d	DD CB d EE							23
SET 5. IY-d	FD CB d EE							
SET 6.A	CB F7							ビットセット
SET 6.B	CB F0							
SET 6.C	CB F1							8
SET 6.D	CB F2							
SET 6.E	CB F3							15
SET 6.H	CB F4							
SET 6.L	CB F5							23
SET 6.HL	CB F6							
SET 6. IX-d	DD CB d F6							23
SET 6. IY-d	FD CB d F6							

ユー・セリット	マシナ語	フ ラ グ 変 化					所 要 クロック サイクル	動 作
		S	Z	H	P/V P V	N		
SET 7.A	CB FF							ビットセット
SET 7.B	CB FE							
SET 7.C	CB F9							ソースの第7ビット ← 1
SET 7.D	CB FA						8	
SET 7.E	CB FB							
SET 7.H	CB FC							
SET 7.L	CB FD							
SET 7.HL	CB FE						15	
SET 7.IX.d	DD CB.d FE						23	
SET 7.IY.d	FD CB.d FE						23	
SLA A	CB 27							シフト・シフト・アリス・シフト
SLA B	CB 20							
SLA C	CB 21							
SLA D	CB 22						8	
SLA E	CB 23							Cy ← 7
SLA H	CB 24							0 ← 0
SLA L	CB 25							
SLA HL	CB 26						15	
SLA IX.d	DD CB.d 26						23	
SLA IY.d	FD CB.d 26						23	
SRA A	CB 2F							シフト・シフト・アリス・シフト
SRA B	CB 28							
SRA C	CB 29							
SRA D	CB 2A						8	
SRA E	CB 2B							Cy ← 7
SRA H	CB 2C							0
SRA L	CB 2D							
SRA HL	CB 2E						15	
SRA IX.d	DD CB.d 2E						23	
SRA IY.d	FD CB.d 2E						23	
SRL A	CB 3F							シフト・シフト・アリス・シフト
SRL B	CB 38							
SRL C	CB 39							
SRL D	CB 3A						8	
SRL E	CB 3B							Cy ← 7
SRL H	CB 3C							0
SRL L	CB 3D							
SRL HL	CB 3E						15	
SRL IX.d	DD CB.d 3E						23	
SRL IY.d	FD CB.d 3E						23	
SUB n	DE.d						7	8ビット引算 (サブトラクト)
SUB A	97							
SUB B	90							
SUB C	91							
SUB D	92						4	A ← A - B
SUB E	93							
SUB H	94							
SUB L	95							
SUB HL	96						7	
SUB IX.d	DD 96.d						19	
SUB IY.d	FD 96.d						19	

ニーモニック	マシン語	フラグ変化						備考 クロック サイクル	動作															
		S	Z	H	P/V P/V	N	C																	
XOR R	BB R							7	換他的論理和(Exclusive-OR)操作。															
XOR A	AB																							
XOR B	AB								A ← A XOR B															
XOR C	AB																							
XOR D	AA							4	<table border="1"><tr><td>a</td><td>b</td><td>c</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	c	0	0	0	0	1	1	1	0	1	1	1	0
a	b	c																						
0	0	0																						
0	1	1																						
1	0	1																						
1	1	0																						
XOR E	AB		0			0	0																	
XOR H	AC																							
XOR L	AD																							
XOR BH	AE							7																
XOR IX=d	DD AE d							19																
XOR IV=d	FD AE d							18																

## フ ラ グ 変 化 表

	命 令	フ ラ グ						コ メ ン ト
		S	Z	H	P/V P V	N	C	
1	8ビット加算 ADD ADC	・	・	・	・	0	・	
2	8ビット減算 SUB SBC CP NEG	・	・	・	・	1	・	
3	論理積 AND	・	・	1	・	0	0	Cフラグを0に するとき
4	論理和 OR XOR	・	・	0	・	0	0	
5	8ビットインクリメント INC	・	・	・	・	0	・	16ビットINC DEC
6	8ビットデクリメント DEC	・	・	・	・	1	・	では変化しない
7	16ビット加算 ADD	・	・	×	・	0	・	
8	16ビットキャリ付加算 ADC	・	・	×	・	0	・	
9	16ビットキャリ付減算 SBC	・	・	×	・	1	・	
10	ローテート・アキュムレータ RLA RLCA RRA RROA	・	・	0	・	0	・	
11	ローテート RL RLC RR RRC	・	・	0	・	0	・	
12	シフト SLA SRA SRL	・	・	0	・	0	・	
13	ローテート・ディジット RLD RRD	・	・	0	・	0	・	
14	10進補正 DAA	・	・	・	・	・	・	
15	ビット反転 CPL	・	・	1	・	1	・	
16	セットキャリ SCF	・	・	0	・	0	1	
17	キャリ反転 CCF	・	・	×	・	0	・	
18	間接アドレッシング入力 IN r, (G)	・	・	0	・	0	・	IN A, (G)では 変化しない
19	ブロック入力出力 INI IND OUTI OUTD	×	×	×	×	×	×	
20	リビート入力出力 INIR INDR OTIR OTDR	×	1	×	×	×	×	
21	ブロック転送 LDI LDD	×	×	0	・	0	・	
22	リビート転送 LDIR LDDR	×	×	0	・	0	0	
23	ブロックサーチ CPI CPD CPIR CPDR	×	×	・	・	1	・	
24	レジスタ, レジスタ LD A, I LD A, R	・	・	0	IFF	0	・	P/Vに IFFが コピーされる
25	ビットテスト BIT	×	・	1	×	0	・	

× 不定

1 1になる

0 0になる

・ 状態にしたがってセット, リセットされる

- 変化せず

IFF: 0のとき割り込み禁止(DI)

1のとき割り込み可(EI)

になっている

LD A, I LD A, R

ではこの値が P/V にコピー  
される

## 付2 Z-80 規格表 (参考)

## (1) CPU

## 絶対最大定格

項 目	記 号	定 格 値	単 位
入 力 電 圧	$V_{IN}$	$-0.3 \sim +7$	V
出 力 電 圧	$V_{OUT}$	$-0.3 \sim +7$	V
動 作 温 度	$T_{opr}$	$0 \sim +70$	°C
保 存 温 度	$T_{stg}$	$-65 \sim +150$	°C

## 電気的特性

## DC特性

(Ta = 0°C ~ +70°C,  $V_{CC} = +5V \pm 5\%$ )

記 号	項 目	最 小 値	最 大 値	単 位	測 定 条 件
$V_{ILC}$	クロック“L”入力電圧	-0.3	0.45	V	
$V_{IHC}$	クロック“H”入力電圧	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
$V_{IL}$	“L”入力電圧	-0.3	0.8	V	
$V_{IH}$	“H”入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	“L”出力電圧		0.4	V	$I_{OL} = 1.8\text{mA}$
$V_{OH}$	“H”出力電圧	≥ 1		V	$I_{OH} = -250\mu\text{A}$
$I_{CC}$	消費電流		150 200	mA	(上段 Z-80 CPU 下段 Z-80A CPU)
$I_{LI}$	入力リーク電流		10	$\mu\text{A}$	$V_{IN} = 0 \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		10	$\mu\text{A}$	$V_{OUT} = 2.4V - V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		-10	$\mu\text{A}$	$V_{OUT} = 0.4V$
$I_{LBI}$	入力時のデータ・バスのリーク電流		±10	$\mu\text{A}$	$0 \leq V_{IN} \leq V_{CC}$

## 端子容量

(Ta = +25°C, f = 1MHz)

記 号	項 目	最大値	単 位	測 定 条 件
$C_{ip}$	クロック入力容量	50	pF	被測定端子以外の手べて の端子は接地
$C_{IN}$	入力容量	8	pF	
$C_{OUT}$	出力容量	12	pF	

## AC 特性

(Ta = 0°C ~ +70°C, VCC = +5V ± 5%)

信号	記号	パラメータ	2-80 CPU 最小値	2-80 CPU 最大値	2-80A CPU 最小値	2-80A CPU 最大値	単位	測定条件
f	t <sub>CL</sub>	クロック周波数	0.4	200	0.25	200	μs	
	t <sub>WH</sub> (φH)	クロック・パルス幅("H")	180		110		ns	
	t <sub>WL</sub> (φL)	クロック・パルス幅("L")	180	200	110	2000	ns	
	t <sub>SETUP</sub>	クロックの立ち上がり・立ち下がり時間		30		30	ns	
A <sub>0</sub> -A <sub>15</sub>	t <sub>OL</sub> (AD)	クロックの立ち上がりから出力までの遅延		145		110	ns	
	t <sub>F</sub> (AD)	出力がフロート状態になるまでの遅延		110		90	ns	
	t <sub>access</sub>	MREQに先立つ出力確立時間 (メモリ・サイクル)	(1)		(1)		ns	
	t <sub>access</sub>	I/O RQ, RDまたはWRに先立つ出力確立時間 (入出力サイクル)	(2)		(2)		ns	C <sub>L</sub> = 50pF
	t <sub>CO</sub>	RD, WR, I/O RQまたはMREQからの出力保持時間	(3)		(3)		ns	
	t <sub>CA</sub>	RDまたはWRからの出力保持時間 (フロート状態への遷移時)	(4)		(4)		ns	
	t <sub>PD</sub> (D)	クロックの立ち下がりから出力までの遅延		230		150	ns	
D <sub>0</sub> -D <sub>7</sub>	t <sub>F</sub> (D)	出力がフロート状態になるまでの遅延 (書き込みサイクル)		90		90	ns	
	t <sub>SD</sub> (D)	クロックの立ち上がりに対するセットアップ時間 (M1サイクル)	50		35		ns	
	t <sub>SH</sub> (D)	クロックの立ち下がりに対するセットアップ時間 (M2 ~ M5サイクル)	60		50		ns	C <sub>L</sub> = 50pF
	t <sub>dem</sub>	WRに先立つ出力確立時間 (メモリ・サイクル)	(5)		(5)		ns	
	t <sub>des</sub>	WRに先立つ出力確立時間 (入出力サイクル)	(6)		(6)		ns	
	t <sub>odf</sub>	WRからの出力保存時間	(7)		(7)		ns	
	t <sub>th</sub>	ホールド時間	0		0		ns	
MREQ	t <sub>HL</sub> (MREQ)	クロックの立ち下がりからMREQ = "L" になるまでの遅延		100		85	ns	
	t <sub>HH</sub> (MREQ)	クロックの立ち上がりからMREQ = "H" になるまでの遅延 (M1サイクル)		100		85	ns	C <sub>L</sub> = 50pF

信号	記号	パラメータ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
MREQ	$t_{\text{MRH}}(\text{MR})$	クロックの立ち上がりからMREQ="H"になるまでの遅延 (M2-M5サイクル)		100		85	ns	$C_L=50\text{pF}$
	$t_{\text{p}}(\text{MR}_L)$	MREQのハルス幅("L")	[8]		[8]		ns	
	$t_{\text{p}}(\text{MR}_H)$	MREQのハルス幅("H")	[9]		[9]		ns	
IORQ	$t_{\text{IOH}}(\text{IR})$	クロックの立ち上がりからIORQ="L"になるまでの遅延 (入出力サイクル)		90		75	ns	$C_L=50\text{pF}$
	$t_{\text{IOH}}(\text{IR})$	クロックの立ち下がりからIORQ="L"になるまでの遅延 (INTAサイクル)		110		90	ns	
	$t_{\text{IOH}}(\text{IR})$	クロックの立ち上がりからIORQ="H"になるまでの遅延 (INTAサイクル)		100		85	ns	
	$t_{\text{IOH}}(\text{OR})$	クロックの立ち下がりからIORQ="H"になるまでの遅延 (入出力サイクル)		110		85	ns	
	$t_{\text{IOH}}(\text{RD})$	クロックの立ち上がりからRD="L"になるまでの遅延 (入出力サイクル)		100		85	ns	
RD	$t_{\text{IOH}}(\text{RD})$	クロックの立ち下がりからRD="L"になるまでの遅延 (メモリ・サイクル)		130		95	ns	$C_L=50\text{pF}$
	$t_{\text{IOH}}(\text{RD})$	クロックの立ち上がりからRD="H"になるまでの遅延 (M1サイクル)		100		85	ns	
	$t_{\text{IOH}}(\text{RD})$	クロックの立ち下がりからRD="H"になるまでの遅延 (M2-M5サイクル)		110		85	ns	
	$t_{\text{IOH}}(\text{WR})$	クロックの立ち上がりからWR="L"になるまでの遅延 (入出力サイクル)		80		65	ns	
WR	$t_{\text{IOH}}(\text{WR})$	クロックの立ち下がりからWR="L"になるまでの遅延 (メモリ・サイクル)		90		80	ns	$C_L=50\text{pF}$
	$t_{\text{IOH}}(\text{WR})$	クロックの立ち下がりからWR="H"になるまでの遅延		100		80	ns	
	$t_{\text{p}}(\text{WR}_L)$	WRのハルス幅("L")	[10]		[10]		ns	
	$t_{\text{p}}(\text{WR}_H)$	WRのハルス幅("H")						
M1	$t_{\text{IOH}}(\text{M})$	クロックの立ち上がりからM1="L"になるまでの遅延		130		100	ns	$C_L=50\text{pF}$
	$t_{\text{IOH}}(\text{M})$	クロックの立ち下がりからM1="H"になるまでの遅延		130		100	ns	
RQ/SH	$t_{\text{IOH}}(\text{RF})$	クロックの立ち上がりからRQ/SH="L"になるまでの遅延		180		130	ns	$C_L=50\text{pF}$



信号	記号	パラメータ	Z-80 CPU		Z-80A CPU		単位	測定条件
			最小値	最大値	最小値	最大値		
$\overline{\text{RFSH}}$	$t_{\text{DR}}(\text{RF})$	クロックの立ち上がりから $\overline{\text{RFSH}} = "H"$ になるまでの遅延		150		120	ns	$C_L = 50 \text{ pF}$
$\overline{\text{WAIT}}$	$t_s(\text{WT})$	クロックの立ち下がりに対するセトアップ時間	70		70		ns	
$\overline{\text{HALT}}$	$t_D(\text{HT})$	クロックの立ち下がりからの遅延		300		300	ns	$C_L = 50 \text{ pF}$
$\overline{\text{INT}}$	$t_s(\text{IT})$	クロックの立ち上がりに対するセトアップ時間	80		80		ns	
$\overline{\text{NMI}}$	$t_s(\text{NMI})$	$\overline{\text{NMI}}$ のパルス幅("L")	80		80		ns	
$\overline{\text{BUSRQ}}$	$t_s(\text{BQ})$	アドレスの立ち上がりに対するセトアップ時間	80		50		ns	
$\overline{\text{BUSAK}}$	$t_{\text{de}}(\text{BA})$	クロックの立ち上がりから $\overline{\text{BUSAK}} = "L"$ になるまでの遅延		120		100	ns	$C_L = 50 \text{ pF}$
	$t_{\text{de}}(\text{BA})$	クロックの立ち下がりから $\overline{\text{BUSAK}} = "H"$ になるまでの遅延		110		100	ns	
$\overline{\text{RESET}}$	$t_s(\text{RS})$	クロックの立ち上がりに対するセトアップ時間	90		60		ns	
	$t_F(\text{C})$	フロート状態になるまでの遅延 ( $\overline{\text{MRQ}}$ , $\overline{\text{IORQ}}$ , $\overline{\text{RD}}$ および $\overline{\text{WR}}$ )		100		80	ns	
	$t_{\text{nr}}$	$\overline{\text{IORQ}}$ に先立つ $\overline{\text{M1}}$ 出力("L")の確立時間 ( $\overline{\text{INTA}}$ サイクル)	111		111		ns	

$$[1] \quad t_{\text{acm}} = t_w(\phi_H) + t_F - 75$$

$$[5] \quad t_{\text{dem}} = t_L - 210$$

$$[9] \quad t_w(\overline{\text{MRQ}}) = t_w(\phi_H) + t_F - 30$$

$$[2] \quad t_{\text{aci}} = t_C - 80$$

$$[6] \quad t_{\text{dei}} = t_w(\phi_L) + t_F - 210$$

$$[10] \quad t_w(\overline{\text{WR}}_1) = t_C - 40$$

$$[3] \quad t_{\text{ca}} = t_w(\phi_L) + t_F - 40$$

$$[7] \quad t_{\text{cdf}} = t_w(\phi_L) + t_F - 80$$

$$[11] \quad t_{\text{mc}} = 2t_C + t_w(\phi_H) + t_F - 80$$

$$[4] \quad t_{\text{cdf}} = t_w(\phi_L) + t_F - 60$$

$$[8] \quad t_w(\overline{\text{RST}}_1) = t_C - 40$$

○データを  $\overline{\text{RD}}$  に同期してバスに送り出すことが望ましい。割り込みアクノリッジ・サイクルでは  $\overline{\text{RD}}$  および  $\overline{\text{RQ}}$  の両方に同期して送り出すことが望ましい。

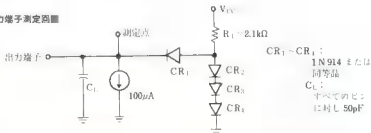
○制御信号はすべて内部で同期がとれているため、クロックについて非同期動作に使用してもよい。

○ $T_A = +70^\circ\text{C}$ ,  $V_{\text{CC}} = +5\text{V} \pm 5\%$  における負荷容量と出力の遅延との関係は次のとおりです。

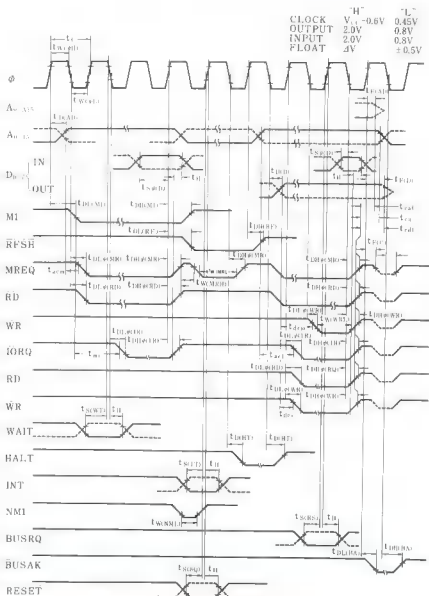
■負荷容量の  $50\text{pF}$  増加につき遅延は  $10\text{ns}$  増加します。負荷容量の最大値は、データ・バスが  $200\text{pF}$  で、他は  $100\text{pF}$  です。

○ $\overline{\text{RESET}}$  の入力幅は最低 3 クロック・サイクルを必要です。

出力端子測定回路



## AC タイミング図



## (2) PIO

## 絶対最大定格

項 目	記 号	定 格 値	単 位
入 力 電 圧	$V_{IN}$	-0.3 ~ +7	V
出 力 電 圧	$V_{OUT}$	-0.3 ~ +7	V
働 作 温 度	$T_{opr}$	0 ~ +70	℃
保 存 温 度	$T_{stg}$	-65 ~ +150	℃

## 電気的特性

## DC 特性

(Ta = 0℃ ~ +70℃,  $V_{CC} = +5V \pm 5\%$ )

記 号	項 目	最 小 値	最 大 値	単 位	測 定 条 件
$V_{ILC}$	クロック“L”入力電圧	-0.3	0.45	V	
$V_{IHC}$	クロック“H”入力電圧	$V_{CC} - 0.6$	$V_{CC} + 0.3$	V	
$V_{IL}$	“L”入力電圧	-0.3	0.8	V	
$V_{IH}$	“H”入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	“L”出力電圧		0.4	V	$I_{OL} = 2mA$
$V_{OH}$	“H”出力電圧	2.4		V	$I_{OH} = -250\mu A$
$I_{CC}$	消費電流		70	mA	
$I_{LI}$	入力リーク電流		10	$\mu A$	$V_{IN} = 0 \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		10	$\mu A$	$V_{OUT} = 2.4V \sim V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		-10	$\mu A$	$V_{OUT} = 0.4V$
$I_{LD}$	入力時のデータ・バスのリーク電流		$\pm 10$	$\mu A$	$0 \leq V_{IN} \leq V_{CC}$
$I_{OHD}$	データ・バス駆動電流	-1.5		mA	$V_{OH} = 1.5V$ ポートBのみ

## 端子容量

(Ta = +25℃, f = 1MHz)

記 号	項 目	最 大 値	単 位	測 定 条 件
$C_{\phi}$	クロック入力容量	12	pF	被測定端子以外のすべての端子は接地
$C_{IN}$	入力容量	7	pF	
$C_{OUT}$	出力容量	10	pF	

## AC 特性

(T<sub>a</sub> = 0°C ~ +70°C, V<sub>CC</sub> = +5V ± 5%)

信号	記号	パラメータ	Z-80 PIO 最小値 最大値	Z-80A PIO 最小値 最大値	単位	測定条件
φ	t <sub>CL</sub>	クロック期間	400 [1]	250 [1]	ns	
	t <sub>WH</sub> (φH)	クロック・パルス幅("H")	170 [1]	105 [1]	ns	
	t <sub>WL</sub> (φL)	クロック・パルス幅("L")	170 [1]	2000 [1]	ns	
	t <sub>su</sub> , t <sub>f</sub>	クロック立ち上がり・立ち下がり時間		30 [3]	ns	
	t <sub>HD</sub>	ホールド時間	11	0	ns	
CE, C/D, B/A	t <sub>su</sub> (CS)	読み出しまたは書き込みサイクルの制御信号のセットアップ時間	280	145	ns	C <sub>L</sub> = 50pF
D <sub>0</sub> - D <sub>7</sub>	t <sub>DR</sub> (D)	RDの立ち下がりからデータ出力までの遅延	430 [2]	380 [2]	ns	
	t <sub>su</sub> (D)	書き込みまたはM1サイクルのデータのセットアップ時間	50	50	ns	
	t <sub>DI</sub> (D)	INTAサイクルのIORQの立ち下がりからデータ出力までの遅延	340 [2]	250 [2]	ns	
	t <sub>F</sub> (D)	RDまたはIORQの立ち上がりから出力バッファ・フロートまでの遅延	160	110	ns	
IEI	t <sub>su</sub> (IEI)	INTAサイクルのIORQの立ち下がりに対するセットアップ時間	140	140	ns	C <sub>L</sub> = 50pF
IEI	t <sub>DR</sub> (IEI)	IEIの立ち上がりからの遅延	210 [4]	160 [4]	ns	
	t <sub>DI</sub> (IEI)	IEIの立ち下がりからの遅延 (注1)	190 [4]	130 [4]	ns	
	t <sub>su</sub> (IEI)	M1の立ち下がりからの遅延 (M1サイクルの直前で割り込みが発生したとき)	300 [4]	190 [4]	ns	
IORQ	t <sub>su</sub> (IR)	読み出しまたは書き込みサイクルのセットアップ時間	250	115	ns	
M1	t <sub>su</sub> (M1)	INTAまたはM1サイクルのセットアップ時間	210	90	ns	
RD	t <sub>su</sub> (RD)	読み出しまたはM1サイクルのセットアップ時間	240	115	ns	
INT	t <sub>su</sub> (IT)	STBの立ち上がりからの遅延	490	440	ns	
	t <sub>DI</sub> (IT)	モード3のときのデータ・バスからの遅延	420	380	ns	

信号	記号	パラメータ	Z-80 PIO 最小値最大値	Z-80A PIO 最小値最大値	単位	測定条件
A <sub>0</sub> -A <sub>7</sub> B <sub>0</sub> -B <sub>7</sub>	$t_{\text{N}}(\text{PD})$	モード1のときの $\overline{\text{STB}}$ の立ち上がりに対するセットアップ時間	260	230	ns	$C_L=50\text{pF}$
	$t_{\text{HS}}(\text{PD})$	モード2のときの $\overline{\text{STB}}$ の立ち下がりに対するセットアップ時間	230 [4]	210 [4]	ns	
	$t_{\text{F}}(\text{PD})$	モード3のときの $\overline{\text{STB}}$ の立ち上がりからポート・バス・フロートまでの遅延	200	180	ns	
	$t_{\text{DR}}(\text{PD})$	モード0のときの書き込みサイクルの $\overline{\text{IORQ}}$ の立ち上がりからポート出力確定までの遅延	200 [4]	180 [4]	ns	
A $\overline{\text{STB}}$ B $\overline{\text{STB}}$	$t_{\text{W}}(\text{ST})$	$\overline{\text{STB}}$ のパルス幅 ("L")	150 [3]	150 [3]	ns	$C_L=50\text{pF}$
A RDY <sub>1</sub> B RDY <sub>1</sub>	$t_{\text{DR}}(\text{RY})$	$\overline{\text{IORQ}}$ の立ち上がりからの応答時間	$t_{\text{C}}+100$ [4]	$t_{\text{C}}+100$ [4]	ns	
	$t_{\text{ER}}(\text{RY})$	$\overline{\text{STB}}$ の立ち上がりからの応答時間	$t_{\text{C}}+100$ [4]	$t_{\text{C}}+100$ [4]	ns	

注 [1]  $t_{\text{C}}=t_{\text{W}}(4\text{-H})+t_{\text{W}}(4\text{-L})+t_{\text{F}}+t_{\text{F}}'$

[2] 負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は200pFです。

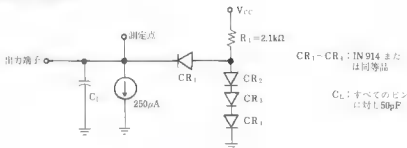
[3] モード2のときは、 $t_{\text{W}}(\text{ST}) > t_{\text{S}}(\text{PD})$ となります。

[4] 負荷容量の10pF増加につき、遅延は2ns増加します。負荷容量の最大値は100pFです。

[注1] デーザー・チェーンがN段ある場合

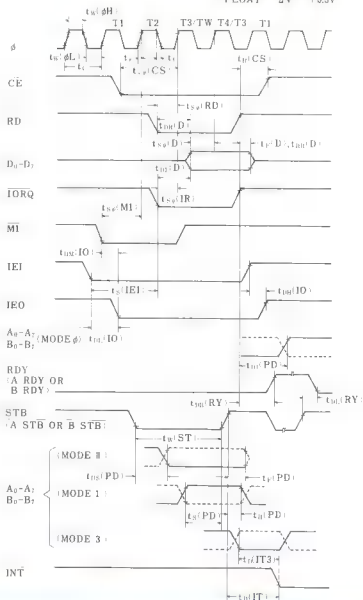
$2.5t_{\text{C}} > (N-2)t_{\text{DR}}(\text{IO}) + t_{\text{DR}}(\text{IO}) + t_{\text{S}}(\text{EI}) + \text{TTLバッファ遅延を働かす必要があります。}$

## 出力端子測定回路



## ACタイミング図

	"H"	"L"
CLOCK	4.2V	0.8V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	4V	0.5V



## (3) CTC

## 絶対最大定格

項 目	記 号	定 格	単 位
入 力 電 圧	$V_{IS}$	$-0.3 \sim +7$	V
出 力 電 圧	$V_{OUT}$	$-0.3 \sim +7$	V
動 作 温 度	$T_{opr}$	$0 \sim +70$	$^{\circ}\text{C}$
保 存 温 度	$T_{stg}$	$-65 \sim +150$	$^{\circ}\text{C}$

## 電気的特性

## DC 特性

(  $T_a = 0^{\circ}\text{C} \sim +70^{\circ}\text{C}$ ,  $V_{CC} = +5\text{V} \pm 5\%$  )

記 号	項 目	最 小 値	最 大 値	単 位	測 定 条 件
$V_{ILC}$	クロック“L”入力電圧	-0.3	0.45	V	
$V_{IHc}$	クロック“H”入力電圧	$V_{CC} - 0.6$	$V_{CC} + 0.3$	V	
$V_{IL}$	“L”入力電圧	-0.3	0.8	V	
$V_{IH}$	“H”入力電圧	2.0	$V_{CC}$	V	
$V_{OL}$	“L”出力電圧		0.4	V	$I_{OL} = 2\text{mA}$
$V_{OH}$	“H”出力電圧	2.4		V	$I_{OH} = -250\mu\text{A}$
$I_{CC}$	消費電流		120	mA	$t_r = 400\text{ns}$
$I_{LI}$	入力リーク電流		10	$\mu\text{A}$	$V_{IS} = 0\text{V} \sim V_{CC}$
$I_{LOH}$	トライステート出力リーク電流		$10^*$	$\mu\text{A}$	$V_{OUT} = 2.4\text{V} \sim V_{CC}$
$I_{LOL}$	トライステート出力リーク電流		$-10^*$	$\mu\text{A}$	$V_{OUT} = 0.4\text{V}$
$I_{OH0}$	ターリントン駆動電流	$-1.5^*$		mA	$V_{OH} = 1.5\text{V}$ $ZC/TO_0 - ZC/TO_2$ に適用

\* 流入電流を正、流出電流を負とします。

## 端子容量

(  $T_a = +25^{\circ}\text{C}$ ,  $f = 1\text{MHz}$  )

記 号	項 目	最大値	単 位	測 定 条 件
$C_{\Phi}$	クロック入力容量	25	pF	視測定端子以外のすべての端子は接地
$C_{IS}$	入 力 容 量	5	pF	
$C_{OUT}$	出 力 容 量	10	pF	

## AC 特性

(T<sub>A</sub> = 0℃ ~ +70℃, V<sub>CC</sub> = +5V ± 5%)

信号	記号	パラメータ	Z-80 CTC		Z-80A CTC		単位	備考
			最小値	最大値	最小値	最大値		
Φ	t <sub>CT</sub>	クロック周期	400	(1)	250	(1)	ns	
	t <sub>PH</sub> (H)	クロック・パルス幅("H")	170	2000	105	2000	ns	
	t <sub>PL</sub> (L)	クロック・パルス幅("L")	170	2000	105	2000	ns	
	t <sub>PLH</sub>	クロック立ち上がり・立ち下がり時間		30		30	ns	
CS/CT	t <sub>SH</sub>	ホールド時間	0		0		ns	
	t <sub>SD</sub> (CS)	読み出し、または書き込みサイクルの制御信号のセ・トアップ時間	160		60		ns	
	t <sub>ACC</sub> (D)	RQの立ち上がりからデータ出力までの遅延		440		300	ns	(2)
	t <sub>OP</sub> (D)	書き込み、またはM1サイクルのセ・トアップ時間	ns		50		ns	
D <sub>0</sub> -D <sub>7</sub>	t <sub>PH</sub> (D)	INTAサイクルのRQの立ち上がりからデータ出力までの遅延		340		200	ns	(2)
	t <sub>PL</sub> (D)	RQの立ち上がりから出力バッファ・アブソートまでの遅延		230		110	ns	
	t <sub>EL</sub> (D)	INTAサイクルのRQの立ち下がりに対するセ・トアップ時間	200		140		ns	
	t <sub>EL</sub> (IO)	IEIの立ち上がりからの遅延		220		160	ns	(3)
IEI	t <sub>EL</sub> (IO)	IEIの立ち下がりからの遅延		190		130	ns	(3)
	t <sub>SH</sub> (IO)	M1の立ち上がりからの遅延(M1サイクルの直前で無効込みを見たとする)		260		190	ns	(3)
	t <sub>SD</sub> (IR)	読み出し、または書き込みサイクルのセ・トアップ時間	250		115		ns	
M1	t <sub>SD</sub> (M1)	INTA、またはM1サイクルのセ・トアップ時間	240		90		ns	
RD	t <sub>SD</sub> (RD)	読み出し、またはM1サイクルのセ・トアップ時間	240		125		ns	
INT	t <sub>DCX</sub> (IT)	CLK/TRGの立ち上がりからの遅延		2t <sub>CL</sub> (Φ) + 200		2t <sub>CL</sub> (Φ) + 140	ns	カウンタ・モード
	t <sub>OP</sub> (IT)	Φの立ち上がりからの遅延		t <sub>CL</sub> (Φ) + 200		2t <sub>CL</sub> (Φ) + 140	ns	タイマ・モード
CLK/TRG	t <sub>CL</sub> (CK)	カウンタ・クロック周期	2t <sub>CL</sub> (Φ)		2t <sub>CL</sub> (Φ)		ns	カウンタ・モード
	t <sub>PL</sub> (CK/TR) / t <sub>PL</sub> (CK/TR)	カウンタ・クロックおよびトリガの立ち上がり・立ち下がり時間		50		30	ns	
	t <sub>SH</sub> (CK)	同期カウンタに要するクロックのセ・トアップ時間	210		130		ns	カウンタ・モード
	t <sub>SH</sub> (TR)	プリスケアラの即時起動に要するトリガのセ・トアップ時間	210		130		ns	タイマ・モード
	t <sub>PH</sub> (CTH)	カウンタ・クロックおよびトリガのパルス幅("H")	200		120		ns	カウンタ・モードおよびタイマ・モード
	t <sub>PH</sub> (CTL)	カウンタ・クロックおよびトリガのパルス幅("L")	200		120		ns	
ZC/TO	t <sub>PH</sub> (ZC)	Φの立ち上がりからZC/TO="H"までの遅延		190		120	ns	カウンタ・モードおよびタイマ・モード
	t <sub>PL</sub> (ZC)	Φの立ち下がりからZC/TO="L"までの遅延		190		120	ns	



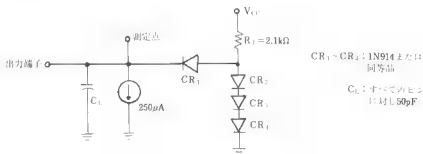
注 [1]  $t_r = t_w(\Phi_L) + t_w(\Phi_H) + t_F + t_f$

[2] 負荷容量の50pF増加につき、遅延は10ns増加します。負荷容量の最大値は、テーク・パスが200pFであり、他は100pFです。

[3] 負荷容量の10pF増加につき遅延は2ns増加します。負荷容量の最大値は100pFです。

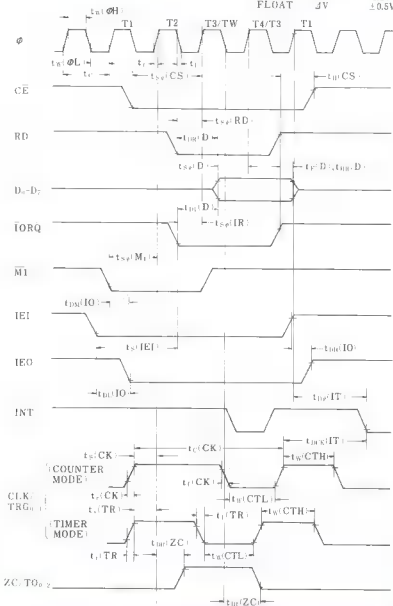
[4] RESETの入力幅は最低3クロック・サイクルが必要です。

## 出力端子測定回路



## AC タイミング図

CLOCK	"H"	"L"
OUTPUT	$V_{OL} = 0.6V$	0.45V
INPUT	2.0V	0.8 V
FLOAT	$\pm 2V$	0.8V
	$\pm 1V$	$\pm 0.5V$



# 索引

## ア行

アウト	60	
アキュムレータ	86	
アセンブラ	36	
アセンブリ言語	36	
アド	98	
アドウィズキャリ	98	
アドレス	26	
アドレスデコーダ	44	
アドレスバス	30	
アンド	100	

イネーブル	44	
インクリメント	98	
インストラクションサイクル	32	
インタラプト	28, 68	
インタラプトイネーブルアウト	80	
インタラプトイネーブルイン	80	
インデックスレジスタ	84, 92	
イン	60	

ウエイト 28, 44, 66

エクスタループアウト	100
エムワン	64

オア	100
オーバーフローフラグ	118
オペコード	32, 36
オペランド	32, 36

## カ行

カウンタ 88

疑似SRAM	22
キャリフラグ	98, 118

クロック	48	
クロックサイクル	48	
交換命令	90	
コメント	36	
コール	116	
コンプリメント	100	
コンプリメントキャリフラグ	118	

## サ行

サインフラグ	118	
サブトラクト	98	
サブトラクトウィズキャリ	98	
サブルーチン	26, 116	
算術演算命令	98	
算術的右シフト	104	

シフト命令	104	
シフトライトアリスメチック	104	
シフトライトロジカル	104	
シフトレフトアリスメチック	104	
ジャンプ	112	
ジャンプリラティブ	112	

スタック	94	
スタックポインタ	84, 94	

正論理	30	
セット	102	
セットキャリフラグ	118	
ゼロフラグ	118	

ソース	36	
ソースプログラム	36	

チップセレクト 64

ディクリメントジャンプノンゼロ	112
デイスティネーション	36
ディスプレイメント	92
デクリメント	98, 100
デージ・チェーン	80
デシマルアジャストアキュムレータ	118, 120
データバス	30
転送命令	96

トライステート	12
---------	----

## ナ 行

ニゲイト	98
ニモニック	36
入出力命令	60

ネスティング	94
--------	----

ノンオペレーション	66
ノンマスカブルインタラプト	68, 70

## ハ 行

排他的論理和	100
バイナリコードデシマル	120
バスアタノリッジ	28, 82
バスリクエスト	28, 82
ハーフキャリフラグ	118
マリティフラグ	118
判定条件付ジャンプ命令	112
ハンドシェイク線	128
汎用レジスタ	88

ビット	24, 102
ビット操作命令	102
ビットパターン	24

フエッチサイクル	28, 54
フッシュ	116
フラグ	86
プログラムカウンタ	84
ブロックサーチ命令	108
ブロック転送命令	108

ブロック入出力命令	108
負論理	30
ヘキサデシマル	38
ベクトル	78
ペリフェラル	12

ポインタ	88
補助レジスタ	90
ポップ	116
ホルト	28, 66

## マ 行

マクロアセンブラ	36
マシン語	26
マシンサイクル	28
マスクROM	22

無条件ジャンプ	112
---------	-----

命令サイクル	32
メインルーチン	26
メモリ空間	42
メモリライトサイクル	28
メモリリクエスト	44, 64
メモリリードサイクル	28, 56

モード0	72
モード1	74
モード2	78

## ラ 行

ライト	64
ラベル	36

リスタート	116
リセット	28, 50, 102
リターン	116
リターンフロムインタラプト	72, 116
リターンフロムノンマスカブルインタラプト	116
リード	64
リフレッシュ	22
リフレッシュサイクル	28

リロケートブル 112

レジスタ 88

ローテート 104

ローテートライト 104

ローテートライトサーキュラ 104

ローテートレフト 104

ローテートレフトサーキュラ 104

ロード 96

論理演算命令 100

論理積 100

論理否定 100

論理和 100

# ワ 行

割り込み 26, 68

割り込み処理ルーチン 26

割り込みモード 0 74

割り込みモード 1 76

割り込みモード 2 78

# 数字、アルファベット

2 進化 10 進数 120

8 ビットレジスタ 88

10 進補正 98

16 ビットレジスタ 88

A レジスタ 84

ADC 98

ADD 98

AND 100

ARDY 15

ASIC 10

ASTB 15, 128

B レジスタ 108

BCD 120

BIT 102

BRDY 15

BSTB 15

BUSAK 82

BUSRQ 82

C フラグ 98

C レジスタ 108

CALL 116

CCF 118

CE 124

CP 98

CPD 161

CPDR 161

CPI 161

CPIR 161

CPL 100

CPU 12

CTC 16

D レジスタ 84

DAA 98, 118

DEC 98, 100

DI 72

DJNZ 112

DMA 18

E レジスタ 84

EEPROM 22

EI 72

EPROM 22

EX 90

EXX 90

F レジスタ 84, 86

H レジスタ 84

HALT 66

I レジスタ 78, 84, 86

IEI 80

IEO 80

IM 0 72

IM 1 72

IM 2 72

IN 60

INC 98

IND 162

INDR 162

INI	162
INIR	162
INT	68
IO 空間	42
IO ライトサイクル	28, 60
IO リクエスト	44, 64
IO リードサイクル	28, 60
IORQ	44, 64
IX レジスタ	92
IY レジスタ	92
JP	112
JR	112
L レジスタ	84
LD	96
LDD	108
LDDR	108
LDI	108
LDIR	108
MI	64
MREQ	44, 64
NEG	98
NMI	68
NOP	66
NVRAM	22
OR	100
OTDR	167
OTIR	167
OUT	60
OUTD	167
OUTI	167
PIO	14
POP	116
PROM	22

PUSH	116
R レジスタ	86
RAM	22
RD	64
RES	102
RESET	50
RET	116
RETI	72, 116
RETN	70
RL	104
RLA	169
RLC	104
RLCA	169
RLD	104
ROM	22
RR	104
RRA	169
RRC	104
RRCA	169
RRD	104
RST	116
SBC	98
SCF	118
SET	102
SIO	20
SLA	104
SRA	104
SRL	104
SUB	98
UVEPROM	22
WAIT	66
WR	64
XOR	100

〈著者略歴〉

横田 英一 (よこた えいいち)


昭和 47 年 東京電機大学工学部

電子工学科卒

現 在 シャープ株式会社

電子部品営業本部

- 本書の内容に関する質問は、オーム社出版部「(書名を明記)」係宛。  
書状またはFAX (03-3293-2824) にてお願いします。お受けできる質問は本書で紹介した内容に限らせていただきます。なお、電話での質問にはお答えできませんので、あらかじめご了承ください。
- 万一、落丁・乱丁の場合は、送料当社負担でお取替いたします。当社販売管理部宛お送りください。
- 本書の一部の複写複製を希望される場合は、本書扉表を参照してください。

 <(株)日本著作出版権管理システム委託出版物>

## 新版 図解 Z-80 の使い方

---

平成 5 年 8 月 20 日 第 1 版第 1 刷発行

平成 15 年 3 月 15 日 第 1 版第 14 刷発行

著 者 横田 英一

発行者 佐藤 政次

発行者 株式会社 オーム 社

郵便番号 101 8460

東京都千代田区神田錦町 3-1

電 話 03(3233)0641 (代表)

URL <http://www.ohmsha.co.jp/>

© 横田英一 1993

印刷 中央印刷 製本 三木舎

ISBN4-274-07759-4 Printed in Japan

# わかる本の のご案内

「わかる本」はいま必要な技術と知識をテーマに、その本質を圖やイラストを用いて、やさしく・わかりやすくをモットーに編集した入門書です。専門学校生・大学生から社会人の方々の必読の書です。

## Javaがわかる本

イー・コミュニケーションズ株式会社 ■  
(A5判・128頁・本体 1500円)

- Javaの概要 / Javaの歴史 / 言語としてのJava / Javaを用いたユーザインタフェース / 組み込み向けのJava / エンタープライズ向けのJava / JavaとXML

## ネットワークがわかる本

石川 裕 ■  
(A5判・132頁・本体 1500円)

- コンピュータシステムとネットワーク / ネットワークの伝送技術 / 交換技術 / プロトコル / LAN / WAN / インターネット / 通信のセキュリティ / これからのネットワーク

## 通信プロトコルがわかる本

石川 裕 ■  
(A5判・142頁・本体 1500円)

- データ通信の基礎 / 下位層のプロトコル / 上位層のプロトコル / LANとTCP/IPのプロトコル / 実際のネットワークとプロトコル / 今後のネットワーク

## インターネットがわかる本

芝野耕司 ■  
(A5判・148頁・本体 1500円)

- インターネットとは / インターネットの現在 / インターネットの仕組み / 電子メールとその拡張 / WWWの仕組み / インターネットの未来

## Webサイトがわかる本

樺山友一 ■  
(A5判・128頁・本体 1500円)

- Webサイトとは / Webサイトの開発プロセス / Webサイトの内部 / セキュリティ方式の決定 / コンピュータ構成の決定 / Webサイトの設計 / Webサイトのテスト / Webサイトの運用

## SGML/XMLがわかる本

芝野耕司 ■  
(A5判・162頁・本体 1500円)

- SGMLへ至る道 / 標準一般化マーク付け言語SGML / SGMLからHTML / そしてXML / 形式適合XML文書 / 正しいXML文書 / XML Names / XMLの事前空間 / XPath / XMLでの部分空間の指定 / XSLT / XMLでのスタイル指定と変換 / XMLをうまく使いこなすために

## データベースがわかる本

鈴木 健司 ■  
(A5判・120頁・本体 1500円)

- データベースの概念 / データモデルの概念 / 関係データモデル / データベースの設計 / データベース言語SQL / 関係DBMSのGUI / データベースの新たな展開

## オブジェクト指向がわかる本

佐藤英人 ■  
(A5判・136頁・本体 1600円)

- オブジェクト指向の基礎 / オブジェクトの実装 / オブジェクト指向プログラミング / オブジェクトの部品化と再利用 / オブジェクトの分析と設計 / オブジェクト指向の開発

## SQLがわかる本

芝野耕司 ■  
(A5判・136頁・本体 1400円)

- リレーショナルデータベースとSQL / 表とビューの定義 / データ定義言語 (DDL) / SQLでの問合せとMSAccessのクエリー / 表の結合 (JOIN) と併合 (UNION) / SQLでのデータ操作 / Accessのフォームとレポート / 副問合せ / データベース設計と正規化 / コーサル処理とホスト言語からの呼び出し / トランザクション管理と安全保護 / SQLの拡張

## データウェアハウスがわかる本

鈴木健司 ■  
(A5判・120頁・本体 1500円)

- データウェアハウスの概念 / データウェアハウスの基本構成 / データウェアハウスの設計と構築 / 多次元データウェアハウス / データマートの設計と構築 / データウェアハウスの活用 / データウェアハウス構築の開発手順

## プログラミングがわかる本

本田 徹夫 ■  
(A5判・152頁・本体 1500円)

- コンピュータ言語とは / コンピュータ言語の第一歩 / BASIC言語でプログラミング / C言語プログラミング / オブジェクト指向プログラミング / Visual Basicで作成 / 応用問題と実力アップ

※上記書籍の表示価格は本体価格です。別途消費税が加算されます。  
※本体価格の変更、品切れが生じ場合もございますので、ご了承ください。  
※実際に商品がない場合または数量に注文の個数と表記列にご連絡ください。

TEL.03-3233-0543 FAX.03-3293-6224



人工知能の主要テーマの考え方が  
簡単な例で理解できる！

# 新しい人工知能

前田 隆  
青木文夫 共著

知識処理をテーマに概要の説明だけでなく、理論や考え方までをわかりやすく解説している。また、Lips(基礎編)、Prolog(発展編)による実際のプログラミングをとおしてアルゴリズムやシミュレーションの手法が身に付くよう工夫されており、各章末の練習問題で学習内容を確認できる。

## 基本編



### 目次

- 1章—人工知能入門
- 2章—問題解決と状態空間
- 3章—問題解決と探索法
- 4章—人工知能プログラミング—Lips言語—
- 5章—記憶モデルと知識表現
- 6章—エキスパートシステムと知識工学

A5判・168頁・本体 2200円

## 発展編



### 目次

- 1章—論理と推論
- 2章—知識処理と論理プログラミング
- 3章—人工知能プログラミング—Prolog言語—
- 4章—不完全知識の処理
- 5章—学習と知識獲得
- 6章—知的エージェントと分散知能

A5判・188頁・本体 2400円

※上記書籍の表示価格は本体価格です。別途消費税が加算されます。

※本体価格の変更、品切れが生じる場合もございますので、ご了承ください。

※書店に商品がない場合または直接ご注文の場合はお記帳にご連絡ください。

TEL.03-3233-0643 FAX.03-3293-6224

B-41006-28

## 好評関連書籍

### C言語によるプログラミング ー基礎編ー 第2版

内田智史 監修 株式会社システム計画研究所 編 B5判 400頁 2200円

### C言語によるプログラミング ー応用編ー

内田智史 編 B5判 390頁 2400円

### C言語によるプログラミング ースーパーリファレンス編ー

内田智史・秋元 勝・北川雅巳・大津 崇 共著 B5判 560頁 2800円

### 新版 入門C言語

三田典玄 著 B5変判 240頁 1600円

### 新版 応用C言語

三田典玄 著 B5変判 304頁 2200円

### 新版 実習C言語

三田典玄 著 B5変判 312頁 1900円

### WebObjectsアプリケーション開発ガイド

George Ruzek 著 テクニカルラボ 訳 B5変判 456頁 5700円

### CodeWarriorではじめるPalm/Visor/CLIE プログラミング

漆畑広樹 著 B5変判 344頁 3200円

### Perlデバッグ明快技法

Martin Brown 著 岡田長治 監訳 B5変判 512頁 3800円

### Javaデバッグ明快技法

Will David Mitchell 著 鈴木義幸 監訳 B5変判 416頁 3800円

### IIS 5 ASP スクリプティングガイド Windows 2000でWebプログラミング

佐藤銀一 著 B5変判 344頁 2200円

### 実用SQL SQL Sever 7 / MSDE対応

佐藤銀一 著 B5変判 280頁 1900円

### IA-64プロセッサ基本講座

池井 満 著 A5判 296頁 2300円

### Notes/Domino APIプログラミング ●

津田義史 著 B5判 560頁 6600円

### コンピュータグラフィックス 理論と実践

James D. Foley・Andries van Dam・Steven K. Feiner・John F. Hughes 共著 佐藤義雄 監訳 B5変判 1284頁 12000円

●は、CD-ROM付き書籍です。

◎上記書籍の表示価格は、本体価格です。別途消費税が加算されます。

◎本体価格の変更、品切れが生じる場合もございますので、ご了承ください。

◎書内に商品がない場合または直営店注文の場合は上記宛にご連絡ください。TEL:03-3233-0843 FAX:03-3293-6224



オーム社/出版局

ISBN4-274-07759-4

C3055 ¥2800E



定価(本体2800円【税別】)

新版 図解

# Z-80の使い方

